Canada $4.25
USA $3.50

FOR THE Amiga Transactor

APRIL 1989: VOLUME 2, ISSUE 1

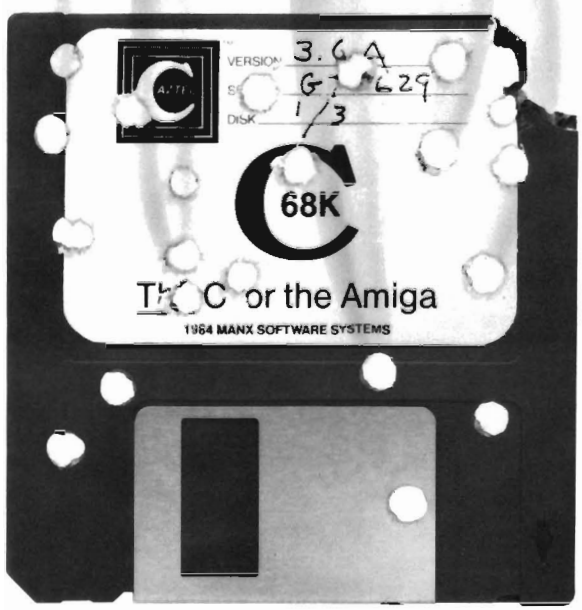
THE MAGAZINE FOR AMIGA PROGRAMMERS

- Assembly language optimization tricks *by John Toebes*
- Language grammars using *Bison/Yacc by Eric Giguere*
- AmigaDOS scripts in 1.3 *by Andy Finkel*
- Starting an Amiga video studio *by Derek Grime*
- Assembly language programming *second in a series by Jim Butterfield*
- ARexx from a user's point of view *by Jay Gross*
- A look at DB and MetaScope *the final 'BreakPoint' article by Vic Wagner*
- DOS file structure under the Fast File System *by Betty Clay*
- Modula-2 string manipulation routines *by Anthony Bryant*
- **Reviews:** Digi-View Gold, excellence!, Anti-Virus, NAG Plus 3.0, Desktop Budget, *Understanding Computers and Cognition*

Dot_Matrix by Derek Grime

0 29074 12122 04

IN SIDE-BY-SIDE COMPARISONS,



LATTICE BLEW MANX AWAY.

Introducing new Lattice C for Amiga DOS, Version 5.0.

Lattice C 5.0 is the fastest compiler by every measure. No other language, no other compiler, gives you faster, more efficient programs. And it now supports 68020 and 68881.

BYTE* Benchmark	Lattice C 5.0	Manx C 3.6	% Difference
dhampstones	42.4	62.7	32%
dhrystones/sec.	1605	1017	58%
matrix manipulation	15.7	50.4	68%
scbigrd	70.2	82.8	15%
sorto	76.4	110.9	31%
sieve	34.4	40.5	15%

And no other compiler gives you a more complete programming environment. This package includes:
• Lattice C Compiler • Compiler Companion Utilities • Source Level Debugger (CodeProbe) • Code Profiler • Global Optimizer • 304 C Library Functions • Blink Overlay Linker • Assembler • Disassembler • Librarian • Sample Files & Examples • Lattice Screen Editor • Installation Program • Comprehensive Documentation.

Our new, easy-to-use, Source Level Debugger, *CodeProbe*, supports both C and Assembly language, and multi-tasking debugging. And the new Global Optimizer will actually *enhance your programs' performance up to 40%.*

So the choice is easy: You can own the fastest, most powerful, most complete C programming environment available. Or for about the same price, you can buy a package that's shot full of holes.

To purchase Lattice C 5.0 or for more information, see your favorite software supplier.

Lattice

Subsidiary of SAS Institute Inc.

Lattice, Incorporated
2500 S. Highland Avenue
Lombard, IL 60148
Phone: 800/533-3577
In Illinois: 312/916-1600

*BYTE is a registered trademark of McGraw-Hill

The Inovatronics Assembly Line:

InovaTools 2

Advanced Assembler Functions

C.A.P.E.68k

Version 2

A Technical Knockout!

Inovatronics understands assembly language. All of our in-house Amiga software development is done in assembler. We know what programmers desire and *require* from an assembler and C.A.P.E.68k delivers: integration, optimization and SPEED!

We also understand that getting started in assembler is difficult for programmers crossing over from a high level language such as "C", or Modula-2, especially on the Amiga! Enter InovaTools 2, a package of expertly programmed routines designed to perform vital, frequently used functions. And, best of all, source code for these routines is provided so that they are not only useful as they are, but also stand as excellent assembler code examples.

CAPE68k Features:

- ARexx macro support
- Optional optimization
- Generates standalone, binary, or BLinkable
- Source size and number of modules unlimited

InovaTools 2 Functions:

- ARexx: Open ARexx port, close ARexx port, wait for ARexx packet, get packet, send ARexx message, reply to ARexx message, extract arguments
- File: Open, close, read from, write to, detemine size (open and closed), load entire file into allocated memory
- Console device: Open device, close it, print char, line, or dec/hex number
- And: Formatted output, library- , linked list-, and memory- management routines, plus math and extensive string functions

The Amiga and the Inovatronics Assembly Line: The only way to *program!*

Inovatronics, Inc.

The REAL POWER in Power Programming
11311 Stemmons Freeway, Suite 8
Dallas, TX 75229 Phone:214/241-9515

Trademarks: Amiga: Commodore International. CAPE68k, InovaTools, Inovatronics: Inovaronics, Inc.

Volume 2, Issue 1

Publisher
Antony Jacobson

Vice-President Operations
Jeannie Lawrence

Assistant Advertising Manager
Julie Cale

Editors
Nick Sullivan
Chris Zamara

Assistant Editor
Malcolm O'Brien

Contributing Writers
Steve Ahlstrom
Ben Blish
Anthony Bryant
Jim Butterfield
Betty Clay
Don Curtis
Matt Dillon
John Faichney
Andy Finkel
Brad Fowles
Eric Giguère
Cathryn Graham
Eric Graham
Tim Grantham
Chris Gray
Derek Grime
Jay Gross
Louis Markoya
Eugene P. Mortimore
Bryce Nesbitt
Hobie Orris
Rob Peck
Larry Phillips
Dan Schein
Steve Simpson
John Toebes
Victor A. Wagner
Clark Williams
James D. Williams
Doug Winger
David Wood

Cover Artist
Derek Grime

for the AMIGA Transactor
The Magazine for Amiga Programmers

BreakPoint Part Five 28
by Victor A. Wagner
A comparison of two debuggers: Manx's DB, and Metadigm's MetaScope.

Quirks And Quandaries 34
by Betty Clay
A collection of unrelated bugs and problems - and what to do about them.

ARexx To You Too 38
by Jay Gross
Programming jargon aside, what is ARexx to the user?

Assembly Language Tricks 42
by John Toebes
When you want really efficient code, every cycle counts - here are some time-shaving techniques!

Shell Scripts Under 1.3 48
by Andy Finkel
In case you wondered why Andy put all that new script stuff into 1.3, the answer is revealed here for the first time - and it talks!

Using The Bison/Yacc Parser Generator 54
by Eric Giguère
Writing a compiler or interpreter is made easier with a parser generator. In the first of a two-part series, Eric talks about grammars.

Programming In M2AMIGA - String Things 58
by Anthony Bryant
Continuing with his exploration of Interface Technologies' Modula-2 compiler, Anthony presents a small library of string handling functions.

Reserve 64
by Nick Sullivan
A short program that lets you test an application under low-memory conditions.

Changes In The Disk Filing System 66
by Betty Clay
Everybody loves the fast file system - but what really makes it tick? Betty investigates...

Departments and Columns

AbsExecBase 4

ViewPort 6

by Larry Phillips

A standard is needed - now - for multiple Amiga serial ports. Larry explains why.

Amiga Dispatches 8

by Don Curtis

Keep a clean machine - but watch out for Murphy!

ACCESS - The best of non-commercial software 14

by Steve Ahlstrom

This month, Steve looks at Zoo, SetCPU, and PathAss.

Mastering The Raster 18

by Derek Grime

Starting off in home video: some secrets you've never heard before.

Assembly Language Programming 22

by Jim Butterfield

Jim 'twists ECHO's tail', and teaches some solid programming theory in the process, of course.

Hard Copy - books, documentation and information 72

by Tim Grantham

Understanding Computers and Cognition: the authors, Fernando Flores and AI guru Terry Winograd, conclude that machines will never be truly "intelligent".

New Mini-Review Section

NAG Plus 3.0 74

excellence! 74

Desktop Budget 75

Anti-Virus 76

Digi-View Gold 76

About the cover: *Dot_Matrix* by Derek Grime

Derek Grime writes the 'Mastering the Raster' column for *TransAmi*, and works full-time at his Amiga-based graphics studio. Here's how he created the picture for this month's cover:

"The picture is 640 by 400 by four bitplanes (16 colours). The original multiple images were animated the old fashioned way - with pencil and paper - and traced into DPaint II with an Easyl drawing tablet. An extreme pose in the sequence was chosen, and rendered in gold hues using DPaint's shade, blend and smooth modes. The multiple background figures were first composited as monochrome images, then run through Pixmate to do an edge detect and add shine effects for a little more interest. Some highlights were also added at this stage. Finally, the two images were put together back in DPaint."

Transactor for the Amiga is published monthly by Croftward Publishing Inc., 10-85 West Wilmot Street, Richmond Hill, Ontario, L4B 1K7. ISSN# 0838-0163. Canadian Second Class Mail Registration No. 7690, Gateway-Mississauga, Ont. USPS Postmasters: send address changes to: Transactor, PO Box 338, Station C, Buffalo, NY, 14209.

Croftward Publishing Inc. is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names are registered trademarks of Commodore Inc.

Subscriptions: see order card at centre of magazine for complete ordering information and prices.

Send all subscriptions to: Transactor Subscriptions Dept. 10-85 West Wilmot Street, Richmond Hill, Ontario, Canada, L4B 1K7, (416) 764-5273. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to our Richmond Hill HQ. For best results, use the postage paid card at the centre of the magazine.

Quantity Orders: In Canada: Ingram Software Ltd., 141 Adesso Drive, Concord, Ontario, L4K 2W7, (416) 738-1700. In the USA: IPD (International Periodical Distributors), 11760-B Sorrento Valley Road, San Diego, California, 92121, (619) 481-5928; ask for Dave Buescher.

Editorial contributions are always welcome. Only original, previously unpublished material will be considered. Program listings and articles, including BITS submissions, of more than a few lines, should be provided on disk. Preferred format is Amiga 3.5-inch diskettes with ASCII text files. Manuscripts should be typewritten, double-spaced, with special characters or formats clearly marked. Photos should be glossy black and white prints. Illustrations should be on white paper with black ink only. IFF picture files or Aegis Draw-Plus structured graphics files are preferred to hardcopy illustrations when possible.

All material accepted becomes the property of Croftward Publishing Inc., except by special arrangement. All material is copyright by Croftward Publishing Inc. Reproduction in any form without permission is in violation of applicable laws. Write to the Richmond Hill address for a writer's guide.

The opinions expressed in contributed articles are not necessarily those of Croftward Publishing Inc. Although accuracy is a major objective, Croftward Publishing Inc. cannot assume liability for errors in articles or programs. Programs listed in Transactor for the Amiga, and/or appearing on Transactor for the Amiga disks, are copyright by Croftward Publishing Inc. and may not be duplicated or distributed without permission.

Production
In-house with Amiga 2000 and Professional Page
Final output by Vellum Print & Graphic Services, Inc., Toronto

Printing
Printed in Canada by Bowne of Canada Inc.

AbsExecBase

TransAmi editorials don't come very regularly, but fortunately, magazines now do; you've probably seen more new issues in the last three months than there were in the entire year before that. Yes, we do intend to keep it up - with the flood of new ideas and products in the Amiga marketplace, even a magazine every month is barely enough.

Look at the NewsLINK section in this issue for a small hint at what's happening: lots of new products by lots of new companies, many entering the Amiga arena from other markets. The more companies we have supporting the Amiga, the better secured the machine becomes, and the more support we can expect for the future. The Catch-22 situation in the computer marketplace - where machine sales are slow until there is software support, which in turn hinges on the number of machines - turns into a snowball effect when the number of machines is large, encouraging more software, which in turn makes the machine more popular. Developers shouldn't look at the increased activity in the market as merely competition, but as a contribution to the snowball effect (or an escape from the Catch-22), which can only benefit everyone in the long run.

Coming back to *TransAmi*, you may have noticed some subtle changes lately. Cosmetically, Volume 2 ushers in our new page layout - no more cute proportional gadget page numbers, but maybe the more utilitarian ones that have taken their place will be easier to locate when you're flipping through the magazine.

As for content, we seem to have acquired a habit of picking up columnists that's hard to break. Jim Butterfield's introductory series on assembly programming, Derek Grime's "Mastering The Raster", covering Amiga professional graphics and video, and Tim Grantham's "HardCopy", on books and documentation, are all new as of last issue. Nor is that the end of our plans in this area - we expect to be able to welcome yet another new columnist to our pages within the next issue or two... but we'll leave his identity as a mystery for you to guess at until then, Edna.

A couple of lines ago we mentioned Tim Grantham as a columnist... we're pleased to say, in *this* paragraph, that Tim will *also* be joining us as a permanent member of the editorial staff, starting with next issue. We were lucky to get Tim: his years of involvement with Amiga journalism and programming make him a rare find - and a perfect choice.

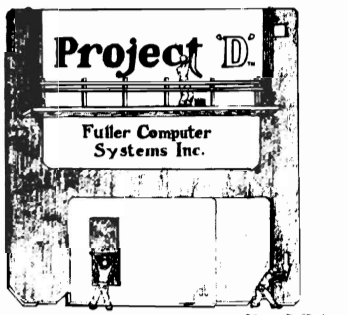
Considering that computer programmers spend a greater portion of their lives communicating, in one form or another, than just about anyone else, it's strange how few letters we get - which is one reason we don't often run a letters column. (The other reason is that we generally tend to underestimate the number of pages we'll need for the articles we just *have* to run.) So how about it? If something you read in *TransAmi* bugs, bewilders, bemuses or otherwise affects you, let us know how you feel. Send letters to: ReplyMsgs, Transactor for the Amiga, 85 West Wilmot Street, Unit 10, Richmond Hill, Ontario, Canada L4B 1K7.

To close out this mixed bag of business, we've been asked to remind our subscribers to renew early to ensure uninterrupted service; renewal notices are still being sent with the last issue of your subscription, but with the transition to monthly service, that doesn't give you much time to get your renewal in before your subscription ends. Until this problem is rectified, see your mailing label to find the last issue in your subscription and renew a couple of issues in advance, just to make sure you don't miss an issue.

At the same time, we would like to thank subscribers and all our customers for being patient with us during the company's recent changes. Some orders got lost or mixed up in the shuffle, and the understanding that most people have shown has really made all the hassle worthwhile.

The Editors

INTRODUCING.....



An Evolution in Disk Utilities for Amiga™ Personal Computers!

FEATURING

- An easy to use, friendly and intuitive user interface.
- A powerful and fast disk backup tool that lets you make backups of your copy-protected Amiga software.
- A disk editing tool that lets you edit raw MFM tracks, AmigaDOS sectors and AmigaDOS files (automatically calculating new checksums).
- A disk cataloging tool that lets you maintain lists of your personal, public domain and commercial software.
- A unique backup tool for duplicating other disk formats including MS-DOS/PC-DOS and Atari ST.
- An easy to read, informative user manual is included.
- This product is not copy-protected in any way.

NOW SHIPPING!

$49.95 EA.

Includes shipping and handling!

Arizona residents add 6.5% sales tax.

TO ORDER

Send check or money order to:
Fuller Computer Systems, Inc.
P.O. Box 9222
Mesa, Arizona 85214-0430
Or CALL (602) 835-5018

Amiga is a trademark of Commodore-Amiga, Inc.

Dealer Inquiries Invited

✓✓✓✓ UEDIT ✓✓✓✓

Programmable Editor For Technical Amiga Users

"Definitely the most sophisticated text editor to come along last year . . . easy for beginners . . . and advanced users . . . reviewing UEDIT is like an art critic reviewing a lump of clay. You can mold it to suit your needs . . ."

Commodore August 1988

PURCHASING OPTIONS (Choose one. Foreign add $3.)

A) $47.00 + $3 shipping, **Uedit** with printed user reference manual

B) $44.00, **Uedit** with user reference manual on disk.

C) $25.00, **Uedit** with command language doc on disk.

Uedit has quietly become the editor of choice for bright programmers and professionals in the USA and more than a dozen countries. Recently it won the Gold Medal in Commodore magazine.

Features: Ultra power, capacity, and flexibility. **Uedit** has ARexx interface, many word processing features, optional spelling checker, and a truckload of editing features. How do you describe a program that lets you add new features when you feel like it? Yet you can use it right away "as is". Edit 20, even 100 or more files. Auto-load/edit/save a batch of files while you relax. Every command and feature is customizable, even the online Help and Teach Keys facility!

Rick Stiles, P.O. Box 666, Washington, IN 47501
(812) 254- 4986

ARexx

The REXX Language for the Amiga

ARexx is a multitasking implementation of the REXX language, an elegant high-level language especially suited for macro-processing. Its clean, simple syntax makes REXX easy to learn for novices — and experienced programmers will appreciate advanced features like compound variables, INTERPRET instruction, and source-level debugging.

- Interactive, Interpreted Operation
- Exceptional String-handling Facilities
- Built-In Library with Over 75 Functions
- Built-In Source-Level Debugger
- Compact, Reentrant code — Only 32K

The Emerging Standard

ARexx is the *de facto* standard for inter-program communication. Software that supports the ARexx command interface can be customized, extended, and combined with other ARexx-compatible software in integrated applications. Watch for announcements of ARexx support from a growing list of Amiga software vendors!

Available Now ... Only $49.95

WShell

The Command Shell You've Always Wanted

WShell is the best command shell on the market, with features and convenience of use that go well beyond the CLI and 1.3 AmigaShell. And yet it's highly compatible with the CLI, so you don't have to learn a new command language to use it!

- Line Editing/Command History/Aliases
- Built-In Commands/Resident Commands
- Prompt String/Window Title "Variables"
- Concurrent Piping
- Transparent Support for ARexx Macros
- Script-bit Support for "Execute" Scripts
- Compact, Reentrant Code — Only 10K

ARexx Command Interface

With WShell you can use REXX-language macro programs like executables — even as filter programs in a piping system. Use the source-level debugging to single-step through a system of concurrently-running "piped" programs!

Available Now ... Only $50

Ask your Amiga dealer for ARexx and WShell, or order them directly from the author. Please include a check or money order plus $2/item shipping ($8 for overseas airmail) and any applicable taxes. Dealer inquiries welcome!

William S. Hawes
P.O. Box 308
Maynard, MA 01754
(508) 568-8695

Amiga is a trademark of Commodore-Amiga, Inc.

ViewPort

The standards dilemma - avoiding the tower of Babel

by Larry Phillips
Copyright © Larry Phillips 1989

Larry Phillips is an Amiga hardware and software hacker from Vancouver, and a sysop on CompuServe's Amiga forums.

If you look around the computer industry, be it in mainframes or micros, you will see standards. There are formal standards, like RS-232 or IEEE-488, and informal or 'de facto' standards, like Xmodem or ARC. There are standards that are effectively in the public domain, and transcend machine types, being available on all sorts of different architectures, as well as those that are proprietary, and available only on one machine type.

The benefits of standards are fairly obvious. Without RS-232, for instance, we would have limited communication between machines, with each manufacturer only being concerned that its own equipment could communicate. We would have a situation wherein a terminal manufacturer would have to decide which communication method to support, and perhaps would have to support multiple methods in order to gain a wide market for its products. Since standards like RS-232 are formally specified, inexpensive to produce, and widely accepted, most companies will opt to provide hardware, firmware and software to implement it, knowing that the rest of the manufacturers will be doing the same, and secure in the knowledge that their products are compatible in at least communication with much of the outside world.

It is not enough that a standard be formally specified in order to be widely used. In fact, there have been many standards formally specified over the years that are little used, if they are used at all.

The reasons for this are varied. Perhaps the hardware is expensive to produce, or perhaps the standard has been outgrown, being superseded by a better standard. In all cases of formal standards, defined by a standards organization, at least some people wanted it enough to lobby for its acceptance, but the standard may even so not be suitable or attractive to most manufacturers.

Sadly, there are cases where the 'NIH' (Not Invented Here) syndrome comes into play. The history of computers is rife with examples of this, and sometimes we are poorer for it. The PAL, SECAM and NTSC television specifications, the BETA and VHS VCR standards, are all examples of the 'NIH' syndrome, and are standards at war, fighting for domination. In the former case, we have seen the results in the adoption of NTSC in North America, and the others in Europe, creating considerable problems for television manufacturers. In the case of BETA versus VHS, we have seen an inferior standard (VHS) outstrip a better one (BETA) by virtue of public acceptance.

Just as the formal specification of a standard does not guarantee its acceptance, the lack of a formal specification, or blessing of a standard by a standards organization, does not doom a de facto standard to oblivion. In fact, many standards that started out as informal methods for certain operations have been widely accepted, and many have become formalized, being accepted by a standards organization, and subsequently published in full detail as to implementation.

Xmodem is a classic case of a standard that became a standard simply because it was there first, doing the job well, and being widely publicized. As a file transfer protocol for CP/M machines, it was all that was required, and came along at a time when it was badly needed. It is almost unheard of to find a terminal program that does not support Xmodem, if it has any protocol transfer capability at all, yet there are definite problems with Xmodem. By modern standards, Xmodem would not even be considered a proper protocol, due to its habit of 'padding' the last block transferred to make up the final 128 byte packet. Despite this shortcoming, and despite the heroic efforts to work around this problem, it is still widely used.

This brings up the other side of the standards coin. Standards can be as much of a curse as they are a blessing. They bring to the industry a sort of inertia, a reluctance to change from a proven, popular standard to a better method. In a typical 'chicken and egg' dilemma, a manufacturer will often opt for the safe route, sticking with an accepted standard, despite having come up with a better one. They cannot be faulted for this, since a move to a different standard could well spell doom for their machine if the other manufacturers do not adopt it in their own products.

In the early days of the Amiga, it was thought that a standard for interchange of files between programs would be a good thing, and IFF was developed to handle it. The first few IFF standards concerned graphics, music, and text, and served their purpose well, especially the graphics specifications. Unlike some other machines, where each software vendor uses their own file format, Amiga paint programs can usually interchange files easily, with no conversion utilities required.

Unfortunately, the IFF standard did not initially cover much more than still pictures. When animation and ray tracing programs started to appear, attempts were made to incorporate their needs into IFF. There was much disagreement as to what formats were most suitable, and to this day we have a number of different file formats, requiring conversion utilities such as *Interchange* from Syndesis.

As I write this, another standard is in danger of being implemented in an ad hoc manner, with all that implies in terms of incompatibility. I am speaking of multiple serial ports, a feature Amiga owners have been looking forward to since the machine was introduced.

Currently, there is no standard published by Commodore to define exactly how multiple serial ports are to be handled. A number of manufacturers are working on multiple serial port boards now, and one, ASDG, through their 'Twin-X' IEEE-959 interface, has already announced that it is shipping a two port module. ASDG's method of accessing the serial ports may or may not be the same as the final standard that Commodore comes out with, which means that ASDG will have to either rewrite their drivers or leave them as is.

Rewriting the driver software would be the 'right' way to handle the situation, provided that the Commodore standard includes all the functionality perceived as necessary by ASDG. Meanwhile, software is being written by terminal and BBS authors to handle ASDG's multiple serial board, and the release of a standard by Commodore may mean a rewrite of this software as well. Other companies working on multiple serial boards are in a similar predicament, the only difference being that they have not yet released their products.

This does not imply that they are sitting on the hardware awaiting the pronouncements from Commodore. You can be sure that they have driver software written, and are having terminal programs modified to suit their boards.

The fastest
Modula-2
Software Development System for
AMIGA
$ 249
Demonstration disk $ 10

Modula-2 is the successor of Pascal. Both languages were invented by Professor Niklaus Wirth. Modula-2 is an all purpose language suited for educational, home and professional use. Modula-2 leads to good programming technique, easy readability and maintainable programs. In fact, a lot of schools and universities have chosen Modula-2 for their programming courses.
M2Amiga is a software development system for Modula-2 based on an extremly fast single pass compiler. It is fully integrated into Amiga Workbench and is easy to use from CLI or Workbench. It runs on all Amiga models with a minimum configuration of 512k RAM and one disk drive.
M2Amiga has been developed specially for the Amiga and therefore optimally supports the features of this unique computer.
- Single pass compiler produces optimized native MC68000 code from standard ASCII files. Full support of latest edition of Modula-2.
- Fastest library calls possible on the Amiga. No intermediate code needed for library calls and parameter passing.
- Supports FFP, 32 bit IEEE and 64 bit IEEE reals formats, you can even mix all three formats within the same program.
- Easy access to MC68000 registers, allows the insertion of inline code.
- Full interface definition to the Amiga's operating system following the official definitions.
- The best runtime system currently available on the Amiga. Opens and closes all used libraries, captures all GURUs and releases all used resources after program crash or termination. Routines for System requesters enable user interface consistent with the Amiga's Intuition.
- The emacs based editor places the cursor on compilation errors and explains them in full text.
- In just a few seconds the linker produces stand-alone applications that can be run from CLI or Workbench. Stand-alone applications do not contain any useless overhead – they are compact and fast.
- All programs can produce icons for their output files to allow effective use of the intuitive Workbench environment. Experience the new feeling of program development by selecting and double clicking icons!
- Also included are a project management tool and other utilities.
- Besides the handy manual, an introductory guide to Modula-2 is included. This documentation allows both the beginner and the advanced user to find any information needed quickly and easily.
- Demo programs show the use of various features of the Amiga.
- For professional program development special utilities are available.
  + Source Level Debugger, the new way of debugging your programs. Inspect all your variables (displayed in their own format), zoom into structures as records or arrays or follow pointer chains by clicking the mouse buttons
  + Object file converter, converts standard Amiga object files into M2Amiga format. This allows linking of routines written in other languages.
  + Linker for devices or libraries. Add power to the Amiga by supplying new libraries and devices. Easy and intuitive programming instead of fooling around with Assembler and Amiga linker are possible now!
  + Modula-2 Amiga Programming System Environment

We have Modula-2 Compilers for HP-UX, IBM/370, PCs (Taylor, M2SDS, JPI), OS-9 and Sun, tomorrow even more!
Texas residents add 8% sales tax. International Orders add $ 20 shipping.

The Modula-2 people:
INTERFACE TECHNOLOGIES
3336 Richmond, Suite 323, Houston, TX 77098-9990 (713) 523 8422
Dealer inquiries welcome 1-800-922-9049

A. + L. Meier-Vogt
Im Späten 23
CH-8906 Bonstetten/ZH
Switzerland
Tel. (41) (1) 700 30 37

Amiga Dispatches

Clean at last - but here comes Murphy!

by Don Curtis
Copyright © 1989, Donald A. Curtis

Don Curtis is a police officer in Denver, Colorado, who has for the last three years been assigned to program design and development along with system design and maintenance of ten AT & T Unix 3B2 computers. In his spare time, Don is an assistant sysop on CompuServe's Amiga forums.

It's curious how discussions about hypothetical situations suddenly become real. I'd had a brief discussion with some of the folks on CompuServe about one of the so-called 'Murphy's Laws'. This particular discussion started when someone said they had fried the $9,000 print head of a particular brand of thermal printer. What he had done was to accidentally turn the power off, then quickly turn it back on. Bye-bye print head! This particular printer was manufactured by a company known for rock solid, heavy duty equipment, so it was quite surprising that this type of failure had occurred. The discussion led to how to build latch circuits and time delays to prevent this type of occurrence, and how cheaply one of those circuits could be built (especially when compared to the price of the print head). That, of course, led to my quoting one of Murphy's laws: "The cost of the equipment is inversely proportional to the cost of the part that will fail and destroy the equipment." In other words, a $5 part will take out a $1,000 piece of equipment, but a $1 part will take out a $50,000 piece of equipment.

So, what's all this got to do with the Amiga? Well, my friends, read on and you'll find out.

How dirty is your computer?

I've had my A2000 for about a year and a half now, and I use it four or five hours a day. My computer room is quite small, in the basement with no window or source of outside air. The basement doesn't have any heat outlets in it, nor does my room. Like most basements, it doesn't get out and out cold down there, but it does get a bit too cool to work comfortably. Because of that, I keep the room closed up and, when I'm working, use a small electric heater to make it comfortable. The room being closed doesn't present a big problem, other than the fact that I smoke. I've tried various solutions to catch the smoke. Of course, the obvious solution is to quit smoking, but short of that, I've never found anything that'll keep the air completely clean. This means that every year or two I clean my computer to get rid of the built-up film and dust.

A few nights ago seemed as good a time as any, especially since my hard drive had developed a little bit of a squeal. The squeal I was hearing was typical of the sound caused by the static eliminator block on the spindle starting to rattle in the holder. It doesn't mean anything is wrong, just that it's quite annoying. Before you think that I got what I deserve because I smoke, let me say that I've had brand new hard drives do this right out of the box. I've had drives that are kept in computer rooms, where the air is well filtered and conditioned and smoking isn't permitted, also start to squeal. I'm sure my smoking doesn't help, but it doesn't cause the noise either; it just happens to some drives.

For those who may not know, a static eliminator block is a small conductive block held against the end of the spindle by a piece of spring copper soldered to the ground plane of the drive. It prevents static build-up on the drive platters. That's a good idea, so you don't want to remove them as I've seen folks do.

Anyhow, hard drive and smoking aside, it's a good idea to keep things clean as far as computers are concerned, so I set to the task.

A word of caution is appropriate here. If you aren't familiar with electronics equipment, you may not want to do this yourself. It is possible to ruin chips with static, you could knock parts out of alignment, use the wrong type of cleaner or lots of other things that can harm your computer. If you're not sure of what you're doing, please take the computer to a dealer and have their tech do the cleaning. Of course they'll charge you for it, but in the long run, it'll be worth the peace of mind you get from knowing the job was done correctly.

The very first thing to do if you have a hard drive is *back up the data and park the heads*! No matter how careful you are, something can go wrong (isn't that also one of Murphy's laws?), and you want to make sure you can restore things.

After that, the first step in the actual cleaning is to disconnect the power cable, and all the other cables attached to the computer. Obviously we don't want to get shocked or have the computer turned on while we're cleaning it. Next we'll take the cover off the computer. Yuck! Look at all that dust build-up by the vents, the mouse and keyboard connectors and any other opening in the front. Two thousand hours of drawing air through the computer will do that. It really doesn't matter how clean your room is, some dust will get sucked in and accumulate around the front of the computer.

You don't want to take an ordinary vacuum cleaner and try to suck up that dust. Vacuum cleaners can generate a *lot* of static! Those little hand-held so-called 'computer vacuum cleaners' work okay, but you've got to make a very slight modification first. Get the little bag damp so the dust doesn't go right through it. Another way to get the dust off is with a can of compressed air. You can get them at almost any camera store but *read the directions* on the can. Many of those cans of compressed air get extremely cold if used for more than a second or so at a time. Blow the dust away from the computer. I use a small paint brush to help clean things up.

The disk drives

Now that we've got the big chunks of dust off, we can proceed with taking the rest of the system apart to get it as clean as possible. I pulled all the cables to the hard drive and floppy drives. A little trick that may help you is to mark the cables with where they go and which end is up. The drive cables (the ribbon style) usually have room on the top of the connectors to put a small strip of masking tape for use as a label. I mark them with the drive number and 'UP' ("DF0: UP" or "HD0: UP", and so on). I also pulled the main power connector from the power supply to the motherboard. You'll need to use a small screwdriver to open the lock on the connector, and please be sure not to tug on the wires, since they may break. I then pulled the four rear screws and two front screws holding the power-supply/drive-bay bracket in place and lifted it off as a single unit.

Next, I pulled the drive mounting plate off the top by removing the four screws on the side. Likewise, I removed the four screws holding the 5.25" drive in place and pulled it out of the bay. Like the front, the drives also had a coating of dust on them. These are best cleaned with a small brush rather than a vacuum or air blast. There's much less chance of doing any damage with a brush as long as you use care. In my case, I had a floppy and a 3.5" hard drive mounted on the plate; I removed the hard drive from the plate (it's attached by four screws from the bottom). I don't remove the 3.5" floppy from the mounting plate as I don't find it necessary. Removing the hard drive allows access to the static eliminator block. Please handle the hard drive very carefully. While the latest models are really quite rugged compared to hard drives available a few years ago, they are still more fragile than most other equipment.

Turning the hard drive over, on the controller board you'll find a small copper plate over a hole in the circuit board. This plate will have a little (usually black) block on it that's held against the spindle. It should be fairly easy to identify. Having located this, I clean it with some electronics cleaner. I use Radio Shack Zero Residue Cleaner (part #44-1014). The brand doesn't matter, though - just make sure the cleaner you use leaves *no* residue and is designed for electronic parts. *Never* use any water-based cleaner, nor any cleaner not designed for electronic equipment. I also spray the circuit board to clean off any built-up film. It doesn't take much spray; don't let any run onto the drive mechanism itself.

While that's usually enough to clean it and quiet things down, I go a step further. However, this next step is *not* recommended by hard drive manufacturers. I've used it for years and not had any problems, but that doesn't mean you won't... so if you are going to try it, remember you do so at your own risk! The extra step I take is to place a tiny drop of light oil on the block. The operative word here is 'tiny'. I use a small oiler to put a single drop on the block, then immediately dab it with a paper towel to soak up the excess. This procedure has worked fine for me, and quieted down drives when nothing else would, but as they say in the car business... your mileage may vary, so again, use it at your own risk. I went through the same procedure with the hard drive I've got attached to the Bridge side.

Next come the two floppies. On my 3.5" floppy (Matsushita) there are two very small screws holding a shield in place. If you remove the screws and lift off the shield, you can get to the heads quite easily. This also makes it easy to brush out any dust accumulated inside the drive. I then take a Q-Tip™-type swab and spray the same cleaner used on the hard drive onto the swab and carefully wipe the heads (one on top and one on the bottom) side to side with the swab. Don't rub front to back; that way you don't have to worry about knocking the heads out of alignment. The chances of doing that are slim, but why take any chances? Side to side works fine and is safe.

The 5.25" drive (mine is a Chinon) is even easier. There's no cover to remove to get to the heads. Follow the same procedure: clean up the dust with a brush, then swab the heads side to side with cleaner. I also go so far as to remove the front face plate and clean it with a heavy duty household cleaner like 409™, but that's purely cosmetic and has no effect on the operation of the drive. I hope it's obvious that you must remove the face plate if you want to clean it with a water-based cleaner. And if you do this, make absolutely sure it's dry before you put it back on the drive.

The circuit boards

Well, we're almost done now. My next-to-last step inside the computer is to pull the expansion cards and clean the contacts

on them. I do this by taking a coarse paper towel and, again, spraying it with the Zero Residue Cleaner and then wiping the towel across the contacts. I don't do much else with them except to brush off any dust that may have accumulated on the boards.

That leaves me with a bare motherboard. With it open like this, it's very easy to brush off any accumulated dust on the board itself. I check all the socketed chips to make sure that they are properly seated. Just a quick push down on them is all that's needed. If they're loose, you'll feel them move down. Don't push too hard, as you could damage the motherboard.

Well, the inside cleaning is done, so let's put the computer back together. I remount the 5.25" drive in its bay, then remount the 3.5" hard drive onto the mount plate. Make sure you've got it seated far enough back to clear the front cover on the computer. Next goes the drive mount plate, with the drives attached, onto the power supply-drive bracket. Now the whole works can go back into the computer. Make sure the tabs on the bracket are lined up properly, then put the six screws back in. On my A2000, the two screws in the front didn't have washers on them, while the four in the back did. I don't know if this is typical, but if it is, you now know which have washers and which don't.

Go ahead and put the cards back in their slots, and remember that the cards should be seated fully in the slot. I can't stress enough how important it is to get the cards fully seated - a card like the BridgeBoard will just give you fits if it's not properly seated. Also, remember that the bracket on the back can become hung up on the brace under the motherboard. As I stated in one of my earlier columns, I bend the bottom of the bracket on the card out a bit, and that makes installation of the cards much easier.

The next step is to hook the cables back up. Make sure the cables are all properly attached to the various drives. If you took my advice about marking them, that should be quite easy. The power cable to the motherboard will snap in when it's properly seated. The power cables for the drives are 'keyed' by their shape for the large plugs. The small floppy plugs are only 'keyed' for up/down but not side to side, so be careful.

Check your work... does everything look right? If so, go ahead and attach the external cables to the rear with the exception of the power cable. The power cable is the very last thing you'll hook up. You can leave the cover off for now, just in case something went wrong.

The keyboard

Now comes the keyboard. You've got to be *very* careful here. You can remove the keycaps before cleaning them if you wish, but I strongly suggest you remove them only if you're having a problem with one of the keys sticking or not working properly. If you clean them while they're on the keyboard, use a *damp* rag with some cleaner on it. Don't have the rag dripping wet, because you don't want to get any liquid into the keyboard itself. Again, something like 409 will work fine. Use a Q-Tip or similar swab to get into the tight areas. This method will work fine if you don't have any sticking keys, but won't help free up a sticky key.

If you have sticky keys, or keys that don't respond to a keystroke, you'll have to remove the keycaps. It's very important to be careful because you can ruin a keyboard if you don't pull the keycaps off properly! The keycaps are held on by latches on both sides of the cap. You can buy a special tool for keycap removal. They're available at many computer stores. An alternative is to use something like a small button hook and to hook the keycap from one side and gently pull up and to the side at the same time. The cap will suddenly pop off. The larger keycaps, such as the space bar, are also held on by a guide wire. It's best to leave these alone. They can be a real bear to put back on if you've never done it before.

When you get the keycap off, what you see depends on the keyboard manufacturer. My particular keyboard is made by HiTek; it has small leaf switches that are held open by the slider mechanism. Pushing down on the slider closes the switch and letting it go opens it back up. The leaf switches are very delicate, so stay away from them! A quick spray with the Zero Residue Cleaner should take care of problems with a sticky key or one that doesn't respond when you strike it.

It is possible to accidentally pull the whole slider mechanism off when trying to pull the keycap. If this happens, don't panic, they're easy enough to put back on if you're careful. As I said, the leaf switches are held apart by the slider mechanism so, if the slider comes off, you can't just push it back in. Take something like a toothpick and put it between the leaves, then lift up on the toothpick to open the switch. Then put the slider onto the spring and slowly push it over the switch making sure you don't catch it on the switch leaves. Once the slider is over the switch, it'll snap into place on the backing plate. Putting the keycaps back on the keys is easy. Just put them over the key oriented properly and push down on them - they'll snap into place.

The mouse and the monitor

Once you've got the keyboard cleaned up, the next piece is the mouse. The only thing I do to the mouse is to clean the ball and rollers inside the mouse. If you turn the mouse over, you'll see the access plate for the ball. It rotates to come off; there's usually an arrow indicating how to turn it, and your manual also explains how to remove the ball. The ball is very easy to clean - a little soap and water and it's done. Just make sure it's completely dry before you put it back in the mouse. While it's out, check the rollers inside the mouse. They should be clean and shiny. There is *no* little rubber band in the middle of the rollers. If you see something that looks like a piece of rubber on the rollers... it's dirt and should be cleaned. If you have a magnifying glass, take a look at the 'rubber band' and then slowly and gently scrape it off. Use a wooden toothpick to do

the scraping. Once the ball and rollers are clean, drop the ball back in place and put the access cover back on.

Go ahead and plug the mouse and keyboard back into the front of the computer. That leaves only the monitor screen left to be cleaned. Please remember that there are things you must *never* do around a monitor! A monitor (color *or* monochrome) has high voltage, up to 30,000 volts, driving the tube, and that can KILL you! *Never* clean a monitor while it's turned on. *Never* use a dripping wet rag to clean a monitor. *Never* spray the screen, or spray a rag with the cleaner and then use the rag to clean the screen. *Never* take the covers off the monitor or poke around inside it. Leave things like that to a qualified repair person. They have the proper tools and knowledge to work safely around those voltages.

To clean the monitor screen, spray a rag with something like Windex™ or 409 cleaner, then wipe the screen until its clean. Again, please be careful not to get any liquid inside the monitor. The screen will clean up quite nicely with small amounts of cleaner.

Okay, now everything is clean! Make sure it's all dry and hook the monitor back up to the computer. Check all the connections. If you still have the cover off the computer, recheck all the cables. Finally, plug the power cord back into the computer and turn it on. If anything abnormal happens, turn the computer off immediately!

Murphy strikes

When I did this... aargh!! The computer didn't turn on! The power LED didn't light, the power supply fan didn't go on, the drives didn't spin up! Oh, no - what had I done wrong? I looked at all the connectors, but didn't see anything immediately obvious. The plug was plugged into the wall socket and the house circuit was okay.

Well, I pulled the power connector to the internal drives and tried to turn the power on again. The fan spun, and the computer started to boot. With no power to the drives, it couldn't do anything, but I was encouraged. I shut the power off and then plugged the hard drive back in. I turned it on and everything spun up properly, but the floppy still wasn't powered, so I still couldn't boot.

I reached behind the floppy and tried to insert the power connector into the floppy's socket. It seemed to fit in several positions! The way my computer room is set up, it's not easy to get behind the computer to look at the floppy power socket, and it's not visible from the top. I grabbed a mirror and looked at the connection, and that's when I found what I had done wrong. Without side shields on the socket, there is nothing to prevent the connector-socket combination from being put on offset side-to-side! You can't put them together upside down, but you can sure put them on offset by a pin or two! That meant that I was running either the 12v or 5v feed directly to ground! No wonder the computer wouldn't start!

I installed the connector properly and turned the computer back on... and it booted and ran just fine.

And that's where Murphy's law comes in. For lack of about a tenth of a cent's worth of plastic to mould in side panels on the socket, I could have ruined a multi-thousand dollar computer.

This isn't Commodore's fault - they buy the drives from various manufacturers and that's the way the drives come. It's the fault of the engineer who designed the 'standard' plug and socket combination used on those drives. It's also my fault for not grabbing that mirror earlier and looking at what I was doing. I put the connector on by feel and thought I had it right... obviously I didn't.

So take a lesson from Murphy and me. No matter how many computers you've taken apart, and no matter how familiar you are with electronics equipment... take the time to make sure you've done everything right. As Murphy's first law says, "If anything can go wrong, it will!"

The cleaning was worth it. It only took a few hours of my time, and the hard drives no longer squeal. The keyboard feels crisp again, and the monitor looks much brighter with that layer of dirt off the screen. I don't have to worry about the accumulation of dust inside the floppy drives and, over all, the computer just looks better - just like when it was new.

Guru's Guide™

to the Commodore Amiga

A collection of advanced programmer's guides written by ***Carl Sassenrath*** – the Architect of the Amiga Multitasking Kernel.

Meditation #1 — Interrupts

The heart of the Amiga Exec. This guide covers the entire Exec interrupt system in 112 detailed pages:

Design Philosophy
General Principles
System Structure
Rules for Use
Issues to Consider
Performance Tips
Common Problems
Hardware Interrupts
Software Interrupts
Exec Functions
Priorities
Decoding Internals
Disabling Interrupts
Sharing with Servers

It also includes many useful examples:

Vertical Blanking
Audio Channels
Software Interrupts
Serial Port
Co-Processor
External Ports (CIA)

To Order send $14.95 plus $1.50 shipping and handling to:
Guru's Guide, P.O. Box 1510, Ukiah, CA 95482
Calif. residents add 6% sales tax. Outside North America, add $4.00.

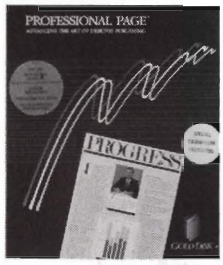
Two week, money-back guarantee if not satisfied.

Guru's Guide is a Trademark of Sassenrath Research

Creative Computers

ORDERS ONLY: 800-872-8882 OUTSIDE CA 213-370-2009 INSIDE CA

VISA, MasterCard, American Express, Discover, International Phone and Mail Orders, Educational, Corporate and Aerospace Purchase Orders Accepted.
4453 Redondo Beach Blvd., Lawndale, CA 90260 Mon.-Sat. 8 a.m.-6 p.m. PST FAX: 213-214-0932



Professional Page ($229)

SOFTWARE

3-D GRAPHICS ... 36.13
3-DEMON ... 71.95
4x4 OFF ROAD RACING ... 32.46
4TH & INCHES CONST. ... 9.95
4TH AND INCHES ... 29.24
64 EMULATOR 2, THE ... 49.95
A B ZOO ... 25.99
A-TALK III ... 64.97
A-TALK PLUS ... 49.95
AAARGH! ... 23.95
AC-BASIC COMPILER V 1.3 ... 134.06
ACCOUNTANT, THE ... 186.95
ADVENTURES OF SINBAD ... 32.45
AEGIS ANIMATOR ... 87.95
AEGIS DRAW 2000 ... 181.95
AEGIS IMAGES-PAINT ... 24.95
AESOP'S FABLES ... 19.95
ALGEBRA I ... 36.95
ALGEBRA II ... 36.95
ALIEN FIRES ... 24.95
ALL ABOUT AMERICA ... 38.95
ALTERNATE REALITY ... 27.95
AMIGA DOS EXPRESS ... 20.95
ANALYTIC ART-GRAPHICS ... 37.48
ANDROMEDA MISSION ... 25.95
ANIMAL KINGDOM ... 32.46
ANIMATE-3D ... 99.95
ANIMATION ... 47.95
ANIMATION EDITOR ... 38.95
ANIMATION MULTIPLANE ... 58.46
ANIMATION STAND ... 32.46
ANIMATOR'S APPRENTICE ... 184.95
ANNALS OF ROME ... 24.95
ARAZOK'S TOMB ... 31.25
ARCADE ACTION PACK ... 34.95
AREXX ... 32.95
ARKANOID ... 19.95
ART GALLERY FANTASY ... 23.36
ART GALLERY I ... 18.73
ART GALLERY II ... 18.73
ASHA'S FONTS ... 58.95
ASSEMPRO ... 71.95
ATREDES (BBS) ... 97.46
• AUDIOMASTER II ... 64.95
AUTODUEL ... 27.00
AWARD MAKER PLUS ... 31.16
AZTEC 68/AM-D ... 224.95
AZTEC C PROFESSIONAL ... 175.46
• AZTEC 68SK C68K SPEC. ... 369.72
B.A.D. DISK OPTIMIZER ... 32.46
BAD CAT ... 30.25
BALANCE OF POWER ... 34.95
BARBARIAN ... 25.95
BARD'S TALE ... 35.95
BARD'S TALE CLUE BOOK ... 10.95
BARD'S TALE II ... 41.95
BARD'S TALE II CLUE BOOK ... 10.50
BASIC GRAMMER SERIES ... 19.46
• BATTLE CHESS ... 32.46
BATTLE SHIP ... 19.46
BBS-PC ... 62.32
BENCHMARK C LIBRARY ... 64.95
BENCHMARK IFF LIBRARY ... 64.95
BENCHMARK MODULA-2 ... 129.95
BENCHMARK SIMPLE LIB ... 64.95
BETTER DEAD THAN ALIEN ... 21.95
BEYOND ZORK ... 33.76
BIONIC COMMANDO ... 25.97
BLACK CAULDRON ... 28.95
BLACK JACK ACADEMY ... 29.95
BLACK LAMP ... 22.00
BLACK SHADOW ... 22.95
BLITZKRIEG (1 MEG VER.) ... 38.95
BLITZKRIEG AT ARDENNES ... 34.95
BLOCKBUSTER ... 32.46
BOBO ... 23.36
BOOMERAID ... 32.46
BREACH ... 25.95
BREACH SCENARIO DISK ... 16.95
BRIDGE 5.0 ... 24.10
BUBBLE GHOST ... 22.71
BUTCHER 2.0 ... 23.13
BUTTON AND BADGE MAKER ... 38.95
C-LIGHT ... 38.95
• C-TAN CASIO CZ ED/LIB ... 126.95
CAD PARTS FOR INTRO CAD ... 16.50
CALCULUS ... 36.95
• CALIFORNIA GAMES ... 32.46
CALLIGRAPHER ... 84.95
CAPE 68K ASSEMBLER 2.0 ... 58.47
CAPITALIZATION SERIES ... 19.46
CAPONE ... 25.95
CAPTAIN BLOOD ... 32.46
CARRIER COMMAND ... 30.95
CASINO FEVER ... 25.95
• CBM DOS 1.3 WORKBENCH ... 24.95
CELEBRITY COOKBOOK, THE ... 22.95
CENTERFOLD SQUARES ... 19.95
CHARON 5 ... 25.96
CHESSMASTER 2000 ... 32.46
CHICKEN LITTLE ... 19.95
CHRONO QUEST ... 33.95
CITY DESK ... 93.95
CITY DESK ART COMPANION ... 19.95
CITY DESK BUNDLE ... 99.95
CLIP ART #1 ... 12.95
CLIP ART #2 ... 12.95
CLIP ART #3 ... 12.95
CLIP ART #4 ... 12.95
CLIP ART #5 ... 12.95
CLIP ART #6 ... 12.95
CLIP ART #7 ... 12.95
CLIP ART #8 ... 12.95
• COMIC FONTS (NEW & NICE!) ... 59.95
COMIC SETTER ... 64.96
COMIC SETTER (FUNNY) ... 22.95
COMIC SETTER (SF DATA) ... 22.95
COMIC SETTER (SUPERHEROS) ... 22.95
COSMIC BOUNCER ... 19.95
CRAPS ACADEMY ... 25.95
CROSSWORD CREATOR ... 34.95
CRYSTAL HAMMER ... 12.96
• CUBEMASTER ... 9.95
• CYBNUS ED PROFFESIONAL ... 84.95
DARK CASTLE ... 25.95
DATA RETRIEVE ... 49.95
■ DATA RETRIEVE. PROF. ... 199.00
DEATH SWORD ... 16.95
DECIMAL DUNGEON ... 32.46
DEEP SPACE ... 19.96
DEFCON 5 ... 25.95
DEFENDER OF THE CROWN ... 34.95
DEJA VU ... 34.95
DELUXE HELP CALLIG. ... 22.71
DELUXE HELP DIGIPAINT ... 21.84
DELUXE HELP DPAINT II ... 21.84
DELUXE HELP PHOTON ... 21.95
DELUXE MAPS ... 16.95
DELUXE MUSIC 2.0 ... 69.95
DELUXE PAINT II ... 89.95
DELUXE PHOTO LAB ... 99.95
DELUXE PRODUCTIONS ... 139.95
DELUXE VIDEO 1.2 ... 69.95
DEMONSTRATOR, THE ... 21.95
DESIGN DISK:
ARCH SCULPT ... 22.74
ARCH VSCAPE ... 22.74
FUTURE SCULPT ... 22.74
FUTURE VSCAPE ... 22.74
HUMAN SCULPT ... 22.74
HUMAN VSCAPE ... 22.74
MICRO8 SCULPT ... 22.74
MICRO8 VSCAPE ... 22.74
DESTROYER ... 25.95
DEVELOPERS TOOLKIT ... 36.95
DEVPAC AMIGA ASSEM ... 129.95
DIGI-PAINT ... 41.95
DIGI-VIEW ... 143.95
DIGI-VIEW UPGRADE ... 13.95
• DIRECTOR'S TOOLKIT ... 24.95
DIRECTOR, THE ... 45.95
DISCOVERY EXP. DISKS ... 12.97
DISCOVERY/MATH ... 25.00
DISCOVERY/SPELL ... 25.00
DISCOVERY/TRIVIA ... 25.00
DISCRETE MATHEMATICS ... 36.95
DISK MECHANIC, THE ... 58.50
DISK TO DISK ... 34.95
DISK WICK ... 32.46
DISKMASTER ... 37.95
DOMINOES ... 16.95
DOS TO DOS ... 37.95
DOUG'S MATH AQUARIUM ... 51.95
DOUGS COLOR COMMANDER ... 19.46
DR. T'S 4-OP ED/LIB ... 92.95
DR. T'S BACH SONGBOOK ... 17.95
DR. T'S CASIO V21 ED/LIB ... 92.95
DR. T'S COPYIST ... 155.38
DR. T'S CZ RIDER ... 92.95
DR. T'S D-100 ED/LIB ... 92.95
DR. T'S D-110 ED/LIB ... 92.95
DR. T'S D-50 ED/LIB ... 92.95
DR. T'S DR. DRUMS ... 17.95
DR. T'S DR. KEYS ... 17.95
DR. T'S DX7 ED/LIB ... 92.95
DR. T'S ESO-APADE ESO-1 ... 92.95
DR. T'S KAWAI K-1 ED/LIB ... 92.95
DR. T'S KAWAI K-5 ED/LIB ... 92.95
DR. T'S KCS ... 154.95
DR. T'S LEXICON ED/LIB ... 92.95
DR. T'S MIDI REC. STUD. ... 43.50
DR. T'S MT-32 ED/LIB ... 92.95
DR. T'S MATRIX-6 ED/LIB ... 92.95
• DRAGON'S LAIR ... 46.50
DREAM ZONE ... 32.46
DRUM STUDIO ... 32.47
• DUNGEON MASTER ... 32.46
DUNGEON MASTER HINTS ... 12.95
DYNAMIC DRUMS ... 49.95
DYNAMIC STUDIO ... 139.00
DYNAMIC-CAD ... 299.00
EARL WEAVER BASEBALL ... 34.95
EASY LOANS ... 24.95
EBON STAR ... 25.96
EMETIC SKIMMER ... 25.95
EMPIRE ... 34.32
ENCORE ... 45.95
EUROPEAN SCENERY DISK ... 17.95
• EXCELLENCE ... 159.00
EXPRESS PAINT ... 90.95
EXTEND ... 25.95
FACC II ... 21.95
FAERY TALE ADVENTURE ... 31.95
FAERY TALE GUIDEBOOK ... 7.95
FALCON ... 32.46
• FAMILY TREE ... 32.46
FANCY 3D FONTS ... 46.95
FANTAVISION ... 42.95
FAST ACCESS WRD. PERF ... 12.95
FERRARI FORMULA ONE ... 33.95
FINAL ASSAULT ... 32.46
FINAL MISSION ... 19.95
FINANCIAL COOKBOOK ... 14.95
FINE PRINT ... 41.95
FIRE 'N FORGET ... 25.95
FIREPOWER ... 16.95
FIRST LETTERS & WORDS ... 33.00
FIRST SHAPES ... 33.00
FISH ... 29.95
FLEET CHECK ... 25.96
FLIGHT SIMULATOR II ... 37.46
FLOW ... 62.32
FONTS AND BORDERS ... 22.95
• FORMS IN FLIGHT II ... 79.95
FORMULA ONE AMIGA ... 19.46
FOUNDATIONS WASTE ... 21.82
FRACTION ACTION ... 32.46
FROST BYTE ... 19.46
GALACTIC INVASION ... 16.95
GALILEO 2.0 ... 45.46
GALLERY 3-D (SCULPT) ... 51.95
GEOMETRIC LIBRARY ... 11.95
GETTYSBURG ... 38.95
GIZMOZ 2.0 ... 39.95
GLOBAL COMMANDER ... 27.46
GOLD DISK FONT SET #1 ... 21.85
GOLD SPELL II ... 29.95
GOLDEN PATH ... 29.95
GOMF! 3.0 (SOFTWARE) ... 27.95
GRABBIT ... 20.95
GRADE MANAGER ... 58.95
GRAND SLAM TENNIS ... 31.25
GRAPHICS STUDIO, THE ... 38.95
GREAT STATES II ... 25.95
• GRETZKY HOCKEY ... 32.46
GUILD OF THIEVES ... 30.95
HARDBALL ... 28.95
HARRIER COMBAT SIM. ... 32.95
HARRIER MISSION ... 16.95
HEAD COACH ... 32.46
HEROES OF THE LANCE ... 27.95
HIT DISK VOL. I ... 32.46
HOLE IN ONE ... 25.99
HOLLYWOOD POKER ... 25.95
HOME BUILDERS CAD ... 129.95
HUNT FOR RED OCTOBER ... 27.46
HYBRIS ... 25.99
ICON PAINT ... 59.95
IMPACT-BUS. GRAPHICS ... 62.46
IMPOSSIBLE MISSION II ... 32.46
INDOOR SPORTS ... 31.22
INOVA TOOLS ... 52.03
INSANITY FIGHT ... 25.95
INSTANT MUSIC 1.2 ... 33.00
INTELLITYPE ... 35.95
INTERCEPTOR ... 37.95
INTERCHANGE ... 29.95
INTERCHANGE CONVERSION ... 16.95
INTERCHANGE OBJECTS #1 ... 16.95
INTERCHANGE TSLV MOD ... 12.96
INTERFONT ... 75.36
INTERNATIONAL SOCCER ... 25.96
INTO THE EAGLE'S NEST ... 28.95
INTROCAD ... 49.95
• INVISION ... 99.95
IT'S ONLY ROCK & ROLL ... 21.95
J FORTH ... 77.96
JET ... 37.46
JET SET FONT SET ... 32.50
JINXTER ... 25.95
JOKER POKER ... 32.46
KAMPFGRUPPE ... 41.95
KARA FONTS ... 54.95
KARATE KID II ... 25.95
KEY TO "C" ... 22.95
KICK-WORK 1.3 ... 22.95
KIDTALK ... 27.95
KIKUGI ... 16.95
KINDERAMA ... 32.46
KING OF CHICAGO ... 34.95
KNIGHT ORC ... 30.95
■ LATTICE "C" 5.0 COMPLETE! ... 199.00
LATTICE C 4.0 ... 162.95
LATTICE C++ ... 357.95
LAZERSCRIPT ... 28.10
LEADER BOARD DUAL PACK ... 22.72
LEADER BOARD TORNA DISK ... 13.95
LEADER BOARD-GOLF GAME ... 27.00
LEARNING CURVE ... 51.95
LEARNING THE ALPHABET ... 19.46
LEATHERNECK ... 25.95
LEISURE SUIT LARRY ... 26.40
LIFE CYCLES AMIGA ... 19.46
LIGHTS! CAMERA! ACTION! ... 59.95
LINKWORD DUTCH ... 20.55
LINKWORD FRENCH ... 20.55
LINKWORD GERMAN ... 20.55
LINKWORD GREEK ... 20.55
LINKWORD ITALIAN ... 20.55
LINKWORD PORTUGUESE ... 20.55
LINKWORD RUSSIAN ... 20.55
LINKWORD SPANISH ... 20.55
LINT (GIMPLE) ... 63.70
LITTLE RED HEN ... 19.95
LOGIC WORKS ADVANCED ... 69.95
LOTTERY MAGIC ... 19.95
• M INTELLIGENT MUSIC ... 143.46
• MAGELLAN ... 119.00
MAGICAL MYTHS ... 32.46
MAGICIAN'S DUNGEON ... 21.95
MAJOR MOTION ... 25.96
MANIAX ... 16.95
MARAUDER II-DUPLICATION ... 27.47
MARBLE MADNESS ... 33.00
MASTERING CLI ... 25.95
MASTERPIECE FONT ... 167.95
MATCH IT ... 25.95
MATH MAGICIAN ... 27.49
MATH TALK ... 31.95
MATH TALK FRACTIONS ... 24.95
MATH WIZARD ... 32.46
MATH-AMATION ... 64.95
MAVIS BEACON TYPING ... 30.89
MAXIPLAN 500 ... 93.95
MAXIPLAN PLUS ... 124.95
MEAN 18 COURSE DISK ... 14.96
MEAN 18 GOLF ... 28.95
MEDIALINE FONTS ... 22.95
MENACE ... 19.98
METACOMCO ASSEMBLER ... 68.95
METACOMCO PASCAL 2.0 ... 92.95
METACOMCO SHELL ... 39.95
METACOMCO TOOLKIT ... 34.95
METASCOPE DEBUGGER ... 59.95
MICROFICHE FILER ... 69.95
MICROFICHE FILER PLUS ... 116.95
MICROLAWYER ... 37.95
MISSION CON-BAT ... 31.16
MODELER 3-D ... 65.00
MOEBIUS ... 39.95
MONEY MENTOR C VERSION ... 59.96
MOVIE SETTER ... 64.95
MULTI-FORTH ... 59.95
MULTI-PREFS ... 19.45
MURAL PRINTER ... 29.95
MUSIC MOUSE ... 51.95
MUSIC STUDENT ... 37.95
NAG PLUS 3.0 ... 49.95
NEWSLETTER FONTS COLOR ... 29.95
OBLITERATOR ... 25.95
OFFSHORE WARRIOR ... 25.96
ONLINE 2.0 ... 43.56
OPERATION CLEAN STREETS ... 25.95
OTG DSM (DISASSEMBLER) ... 43.95
P.O.W. ... 25.95
PAGE FLIPPER PLUS F/X ... 103.95
PAGESETTER ... 93.72
PALADIN ... 25.95
PALADIN (QUEST DISK 1) ... 16.86
PAWN, THE ... 16.21
PCLO PRNT. CUR. LAYOUT ... 199.95
PCLO+ PRNT. CUR. LAYOUT ... 399.95
PHANTASIE ... 27.47
PHANTASIE 3 ... 24.95
• PHASAR-FIN'L MGMT ... 62.48
• PHOTON CELL ANIMATOR ... 89.95
PHOTON PAINT ... 64.96
PHOTON PAINT EXP. DISK ... 19.95
PHOTOSYNTHESIS ... 97.95
PINBALL 1.0 ... 19.46
PINK PANTHER ... 26.95
PIONEER PLAGUE ... 25.96
PIXMATE ... 45.95
PLANET PROBE ... 19.95
PLATOON ... 29.21
PORTS OF CALL ... 29.95
POWER WINDOWS 2.5 ... 58.47
POWERSTYX ... 22.95
PRE CALCULUS ... 36.95
PRESIDENT IS MISSING ... 25.95
PRIME TIME ... 25.95
PRINTMASTER PLUS ... 31.23
PRO MIDI STUDIO ... 130.95
PRO VIDEO FONT SET #1 ... 84.95
PRO VIDEO FONT SET #2 ... 84.95
PRO VIDEO FONT SET #3 ... 84.95
• PRO VIDEO PLUS ... 184.95
PROBABILITY THEORY ... 36.95
• PROFESSIONAL PAGE ... 229.00
PROJECT D ... 31.95
PROSCRIPT ... 32.95
PROWRITE 2.0 ... 78.10
PUBLISHER PLUS ... 99.95
PUNCTUATION SERIES ... 19.46
QUADRALIEN ... 23.61
QUARTERBACK ... 45.95
QUESTRON II ... 32.46
QUIZ MASTER ... 49.95
R.R. AESOP'S FABLES ... 19.95
RAW COPY 1.3 ... 38.95
REACH FOR THE STARS ... 27.46
READ & RHYME ... 31.95
READ-A-RAMA ... 32.46
REALM OF THE WARLOCK ... 32.74
REBEL CHRG. CHICKAMANGA ... 39.95
• REEL FISHIN'—GONE FISHIN' ... 30.88
RETURN TO ATLANTIS ... 34.95
ROAD RAIDERS ... 25.96
ROADWAR 2000 ... 27.47
ROADWAR EUROPA ... 29.95
ROADWARS ... 23.95
ROB. READER UGLY DUCK ... 19.45
ROBBEARY ... 16.21
ROCK CHALLENGE ... 25.95
ROCKET RANGER ... 32.46
ROCKFORD ... 23.95
ROMANTIC ENCOUNTER ... 25.95
SCENERY DSK #7 E. COAST ... 16.71

AMIGA MUSIC PACK

- SHS-10 YAMAHA KEYBOARD
- MIDI INTERFACE
- DELUXE MUSIC INSTRUCTION SET
- "IT'S ONLY ROCK & ROLL" SOFTWARE

ALL FOR A LOW PRICE OF $179! (with purchase of computer $149)

We carry OVER 1000 items
The best NEW products
The LOWEST PRICES

NEW PRODUCTS ARRIVE DAILY.
CALL FOR UNLISTED ITEMS.

Excellence! ($159)

GVP GREAT VALLEY PRODUCTS

For Amiga 2000:
IMPACT HC/45—48 MB 28ms Hard Card ... 799.00
IMPACT HC/40Q—42 MB 11ms Hard Card ... 799.00
IMPACT HC/80Q—84 MB 11ms Hard Card ... 1199.00
For Amiga 500:
IMPACT A500-SCSI/HD20—20 MB HD ... 749.00
IMPACT A500-SCSI/HD45—45 MB HD ... 995.00
IMPACT A500-SCSI/HD40Q—42 MB 11ms High-Speed Hard Drive ... 1049.00
IMPACT A500-SCSI/HD80Q—84 MB 11ms High-Speed Hard Drive ... 1449.00
Quantum 80S, 84 MB 11ms 3.5" HD ... 995.00
Shock mounted, 64KB cache (for 11ms speed), SCSI interface. Compatible with IMPACT or A2090 boards.
Quantum 40S—42 MB version of above drive ... 599.00

OTHER CREATIVE PROMOTIONS

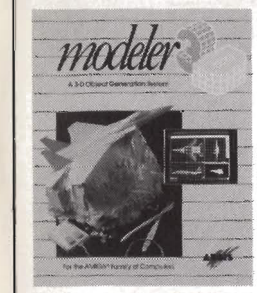
NEW AMIGA 2500 COMPUTER AND 2620 TURBO CARD!
CALL FOR OUR LOWEST PRICE.

AMIGA 500 and 2000 CPU's and package deals available ... CALL FOR DETAILS!

- WordPerfect: The best non-graphic wordprocessor on the Amiga. $399 retail. Our price? **ONLY $179!!!**
- FREE VCR with purchase of computer—call for details.
- Package pricing on computers available—call for details.
- Amigo Live! 2000: **$349**
- Flicker Fixer! (A2000 only) — *Eliminate all flicker and scanline gaps 100% — optional multisync monitor:* **$479**

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL.)

SCENERY DSK #11 E COAST 18.72
SCHEMATIC CAPTURE 199.95
SCRABBLE 27.46
SCRIBBLE 62.32
SCULPT 3-D 89.95
SCULPT 4-D 399.95
■ SDI CINAMAWARE SERIES 34.95
SECONDS OUT 19.46
SECURITIES ANALYST 49.95
SERIAL EXPANDER 15.95
SEX VIXENS 25.95
SHADOW GATE 31.95
SHAKESPEARE 146.95
SHANGHAI-STRATEGY GAME 27.46
SILENT SERVICE 25.95
SINBAD & FALCON 34.95
SKY BLASTER 19.95



AEGIS Modeler 3D ($65)

SKY CHASE 25.95
SKYFOX II 27.45
SLAYGON 19.95
SMOOTH TALKER 33.95
SNAKE PIT 24.95
SOCIAL STUDIES 19.46
SOFTWOOD FILE SG 78.21
SONIX (AEGIS) 49.95
SOUND OASIS 64.95
SOUNDLAB 209.95
SOUNDQUEST:
CZ MASTER 92.95
D-10 MASTER 92.95
D-50 MASTER 92.95
D-110 MAST 92.95
DX7 MASTER 188.95
DX7-II MASTER 188.95
ESQ/ESQ-80 MST 92.95
GENERIC MASTER 77.95
K-1 MASTER 92.95
M-1 MASTER 92.95
MT-32 MASTER 92.95
TEXTURE 92.95
TX-802 MASTER 92.95
TX-81Z MASTER 92.95
SOUNDSCAPE PATTERN SPLAT 35.95
SOUNDSCAPE UTILITIES I 35.95
SOUNDSCAPE UTILITIES II 42.95

TALES ARABIAN NIGHTS 32.46
TALKING ANIMATOR, THE 39.00
TALKING COLORING BOOK 16.95
TANGLE WOOD 25.95
TARGIS 28.95
TECHNOCOP 32.46
TELE EPIC 25.96
TELE WARS II 25.96
TELEGAMES 22.05
TELEWARS 24.95
TERA QUESTRON 25.95
TERRAIN 19.95
TERRORPODS 25.95
TEST DRIVE 34.95
TETRA QUEST 25.95
TETRIS 21.80
TEXTCRAFT PLUS 64.93
THEXDER 23.95
THREE LITTLE PIGS 19.95
THREE STOOGES 35.95
THUNDERBDY 22.95
TIME & MAGIK 26.95
TIME BANDITS 21.95
TOOL CADDY 32.46
TOWER TOPPLER 32.46
TRANSCONT. RAILROAD 25.95
TRIGONOMETRY 36.95
TRUE BASIC 68.95
TURBO PRINT 32.95
● TURBO SILVER 129.95
TV SHOW 64.95
● TV SPORTS FOOTBALL 34.95
TV TEXT 59.32
TXEO PLUS 51.95
ULTIMA III 28.95
ULTIMA IV 38.96
ULTRA DOS 37.95
UMS CIVIL WAR DATA DISK 12.95
UMS VIETNAM DATA DISK 12.95
UNINVITED 34.95
UNIVERSAL MILITARY SIM. 32.46
VAMPIRE'S EMPIRE 29.95
● VIDEO EFFECTS 3D 129.96
VIDEO VEGAS 34.10
VIDEO VISIONS 1 ..... 12.95
VIDEO VISIONS 2 ..... 12.95
● VIDEOSCAPE 3D 124.95
VIDEOTITLER 99.95
VIP VIRUS 32.46
VIRUS (GAMEL) 19.46
VOCABULARY IMPROVE 25.95
W SHELL 31.95
WARLOCK 22.71
WBEXTRAS 24.95
WHO FRAMED ROGER RAB 29.95
WINDOW PRINT II 22.95
WINTER CHALLENGE 9.95
WINTER GAMES 27.46
WORD MASTER 32.46
WORD PERFECT 179.00
WORD PERFECT LIBRARY 84.95
WORKS, THE 124.95
WORLD CLASS LEADER BRD 25.97
WORLD GAMES 27.46
WORLD TOUR GOLF 27.45
WRITE 'N FILE 59.95
WSHELL 32.46
X-CAD 389.00
ZAK MCKRACKEN 29.95
ZERO GRAVITY 19.46

CSA FASTRACK 68020 ACCEL 905.45
DELUXE MIDI INTERFACE 59.95
DESKJET 128K RAM CARD 117.95
● DIGI VIEW GOLD A500/A2000 ..... 149.00
EASYL 1000 TABLET 369.00
EASYL 2000 TABLET 369.00
EASYL 500 TABLET 369.00
EDE MIDI 1000 ..... 48.95
EDE MIDI 500/2000 ..... 48.95

Boing! Mouse

Professional Optical 3-button mouse for your Amiga. Available EXCLUSIVELY at Creative Computers!

The first and only Optical Mouse for the Amiga — $114
This mouse is definitely the best mouse you can buy for your Amiga.

- Optical technology eliminates friction and momentum coefficients for super-precise handling.
- No moving parts for increased reliability.
- Middle button makes it the only mouse compatible with A2024 software and X Windows.
- Compatible with ALL Amiga computers.

ONLY at CREATIVE COMPUTERS

EXP 1000 1M UNPOPULATED 200.00
● FLICKER FIXER (HARDWARE) 479.00
FLICKER MASTER 519.00
GEN ONE GENLOCK 739.00
GOMF 3.0 W/BUTTON 48.95
GVP A2000 AUTOBOOT EPROM 29.95
GVP A500 RAM/ROM MODULE 69.95
HURRICANE 68030 ADAP. BRD 300.00
HURRICANE A1000 ACCEL BRD 350.00
HURRICANE A1000 MEM. BRD 540.00
HURRICANE A2000 ACCEL BRD 500.00
HURRICANE A2000 MEM. BRD 540.00
HURRICANE ACCEL/MEM. BRD 1060.00
HURRICANE DRAM, 1 MEG 525.00
IMPRINT FOR POLAROID PAL 399.00
INTRUDER ALERT 124.02
LASERJET-2 RAM, 1 MEGBYTE 419.95
LENS, 16 MM, NO IRIS F1.6 ..... 21.85
LENS, 16 MM, W/IRIS F1.6 ..... 66.25
LIGHT PEN 97.95
LIVE! 500 ..... 289.00
LIVE! 1000 ..... 270.00
● LIVE! 2000 ..... 349.00
MAGNI 4004 GENLOCK 1595.00
MICROBT. 4 x POP-SIMM BRD. 74.50
MICROBT. 8-UP RAM BRD. 0K 145.00
MICROBT. HARDFRAME SCSI 239.00
MICRON CAGE FOR A500 ..... 199.95
MIDI GOLD 2000 (INT) 68.47
MINISCRIBE 20 MB, 40 MS 295.00
MINISCRIBE 40 MB, 28 MS 615.00
NEC P2200 PRINTER 400.00
NERIKI GENLOCK 1975.00
OKIMATE 20 W/PLUG N PRNT 249.00
OKIMATE PLUG N PRINT 79.95
OVERDRIVE HD CONTROLLER 199.95
PANASONIC WV1410 CAMERA 225.00
PANASONIC WV1500 CAMERA 320.00
PEOPLE METER BIO FEEDBACK 39.95
PERFECT SOUND DIGITIZER 67.95
● PERFECT VISION 169.95
PRO SOUND DESIGNER 119.95
PROCESSOR ACCEL. CMI 159.00
PROGEN 379.00
QUANTUM 80 MB DRIVE, TTMS 995.00
● ROLAND MT-32 TONE MODULE 499.00
SPIRIT 0 MB FOR A1000 ..... 249.00
SPIRIT 0 MB FOR A500 ..... 199.00
STAR NX1000 PRINTER 199.00
STAR NX1000 RAINBOW 279.95
STARBOARD 2 SCSI MODULE 100.00
STARBOARD 2 UPPER DECK 80.00
STARBOARD 2/A1000 0K 250.00
STARBOARD 2/A500 0K 300.00
STARBOARD SB2000 ADAPTER 42.00
STEREO SOUND SAMPLER STRY 87.95
SUBSYSTEM 500 ..... 200.00
■ SUPERGEN 649.00
● SUPRA 2400 MODEM 149.00
SUPRA DRIVE 20 MEG A1000 ..... 699.00
SUPRA DRIVE 20 MEG A500 ..... 699.00
SUPRA DRIVE 30 MEG A1000 ..... 799.00
SUPRA DRIVE 30 MEG A500 ..... 799.00
TRUMP CARD SCSI CNTR 2000 ..... 168.96
TWIN-DRIVES EXT 3.5" FDD 279.00
UNIDRIVE 3.5" EXT. DRIVE 159.00
VI 2000 VIDEO INTERFACE 69.95
VI 2000 VIDEO/RF MODULAT. 79.95
VI 500 VIDEO INTERFACE 59.95
VI 500 VIDEO/RF MODULATOR 69.95
X-SPECS 3D 99.95
XEROX 4020 INK JET COLOR 1140.00
XEROX 4020 STARTER KIT 156.00

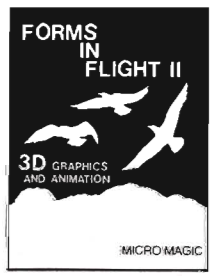
ACCESSORIES

A1084 CARRING CASE 51.95
A500 & MONITOR CARRY BAGS 69.95
AMIGA BAG 29.95
AMIGA BRASS KEY RING 3.95
AMIGA FASHION GOLF SHIRT 19.95
AMIGA FASHION JACKET 39.50
AMIGA MUG 5.78
AMIGA NOTE PADS 4.99
AMIGA PEN & PENCIL SET 19.95
AMIGA PLASTIC VISOR 1.95
AMIGA SLEEVE TSHIRT 7.95

AMIGA T-SHIRT 7.95
AMIGA TERRY CLOTH VISOR 5.50
AMIGA TOWEL 15.95
AMIGA TWILL CAP W/ CORD 7.95
BONUS DISKETTES, VERBATIM 13.95
CABLE, 10' CAMERA 12.95
CABLE, 6' MIDI LINK 7.95
CABLE, AMIGA TO NEC II 24.95
CABLE, AMIGA TO SONY 1302 ..... 19.95
CABLE, AMIGEN TO 1084 ..... 19.95
CABLE, CENTR. TO CENTR 19.95
CABLE, DB25M-DB25F 8 FT. 16.95
CABLE, DRIVE EXT. 2 FT. 19.95
CABLE, DRIVE EXT. 3 FT. 20.95
CABLE, FLCK FIX TO NEC II 19.95
CABLE, FLCK FIX-SONY 1302 ..... 19.95
CABLE, IMAGEWRITER A1000 ..... 22.95
CABLE, IMAGEWRITER A2000 ..... 22.95
CABLE, MONITOR EXT 4FT 21.95
CABLE, MOUSE EXTENSION 9.95
CABLE, PARALLEL A1000 ..... 19.00
CABLE, PARALLEL A500/2000 ..... 19.00
CABLE, SERIAL A1000 ..... 19.00
CABLE, SERIAL A500/A2000 ..... 19.00
CLEANING KIT (ALLSOP) 14.95
CLEANING KIT SM. ALLSOP 6.95
CONTROL CENTRE FOR A500 ..... 71.95
COPY/CAMERA STAND LIGHTS 64.95
CURTIS 3.5" DISK FILE 30 ..... 7.95
CURTIS DIAMOND 33.95
CURTIS DIAMOND PLUS 46.95
CURTIS EMERALD 40.95
CURTIS PHONE LINE SURGE 12.36
CURTIS RUBY 58.95
CURTIS SAPPHIRE 56.95
DESKJET BLACK CARTRIDGE 15.95
DESKJET COURIER FONT CRTG 62.95
DESKJET DUST COVER 29.95
DESKJET EPSON FX-80 EMUL. 62.95
DESKJET HELV 12 POINT 62.95
DESKJET HELV 14 POINT 62.95
DESKJET HELV 8 & 10 PT 79.95
DESKJET HELV ASCII FONTS 104.95
DESKJET LANDSCAPE FONT 79.95
DESKJET LANDSCAPE UPGRADE 79.95
DESKJET LETTER GOTHIC 79.95
DESKJET PRESENTATION FONT 79.95
DESKJET PRESTIGE ELITE 79.95

AmiGen ($149)

DESKJET RMN/HELV SOFT FNT 79.95
DESKJET STAND/ORGANIZER 84.95
DESKJET TMS RMN 12 POINT 62.95
DESKJET TMS RMN 14 POINT 62.95
DESKJET TMS RMN 8 & 10 PT 79.95
DESKJET TMS RMN ASCII 104.95
DIGI-DROID 69.95
DISK FILE 80 ..... 19.95
DUST COVER FOR A2000 ..... 11.95
DUST COVER FOR A500 ..... 8.95
EPYX 500XJ JOYSTICK 14.95
ERGO STICK 17.95
FLICKER MASTER 13.95
FOUR PLAYER ADAPTER 11.95
FUJI DISKETTES 19.95
G/E SURGE PROTECTORS 34.95
GAME PLAY COMPACT DISK 12.95
GENDER BENDER (DIGI-VIEW) 20.95
JOYSTICK Y ADAPTER 10.95
KEYBOARD SKIN A2000 ..... 19.46
KEYBOARD SKIN A500 ..... 19.46
KEYBOARD SKIN A1000 ..... 19.46
LASERJET-2 TONER CARTRIDGE 104.60
LASERJET-2 8.5 x 14 TRAY 104.95
LASERJET-2 COURIER FONTS 144.50
LASERJET-2 ENVELOPE TRAY 79.95
LASERJET-2D ENVELOPE TRAY 279.95
LIGHT GUN 48.95
MICHTRON DISK FOLDER 10.95
MIDI MAGIC 97.95
MIND LIGHT 7 ..... 164.95



Forms In Flight II ($79.50)

MOUSE MATS 6.95
MOUSE TOP 9.95
PAINTJET BLACK CARTRIDGE 23.60
PAINTJET COLOR CARTRIDGE 29.10
PAINTJET CUT-SHEET PAPER 19.10
PAINTJET DUST COVER 19.95
PAINTJET PRINTER STAND 66.95
PAINTJET TRANSPARENCY FILM 54.05
PAINTJET 2-FOLD PAPER 14.95
POWERPLAYERS JOYSTICK 17.95
● RAM, 1 MB (256K-15 CHIPS) 352.00
RAM, 1 MEG x 1 -10 CHIPS 32.50
RAM, 1.5MB (256-15 CHIPS) 528.00
RAM, 1MB (1 MEG-10 CHIPS) 330.00
RAM, 256K x 1 -12 CHIPS 12.00
RAM, 256K x 1 -15 CHIPS 11.00
RAM, 512KB (256-12 CHIPS) 192.00
RAM, 512KB (256-15 CHIPS) 176.00
SONY DISKS 10 PACK 20.95

SPEED MOUSE PAD 9.71
TEAK-TECH 120 DISKHOLDER 32.95
TEAK-TECH 60 DISKHOLDER 27.95
TEAK-TECH 64 DISKHOLDER 24.95

TERMINATOR (JOYSTICK) 15.95
TRACKBALL EMULATES MOUSE 39.95
TRANSPORT CONTROLLER PRO 194.95
XEROX 4020 4-PACK INK 19.95

HOT NEW PRODUCTS
(As featured in this magazine)

VIDEO & AUDIO:
- ■ MidiVU
- ■ C-Zor
- ■ Comic Fonts
- ■ Pro-Sound Designer
- ■ MIDI Magic
- ■ Digi View Gold
- ■ Zoetrope
- ■ Photon Cell Animator
- ■ Sculpt-Animate 4D

GAMES:
- ■ Operation Wolf
- ■ Buble Bobble
- ■ Roston
- ■ Dragon's Loir
- ■ TV Sports Football
- ■ Lords of the Rising Sun
- ■ Cosmic Bouncer
- ■ Rock Challenge
- ■ Univeral Military Simulator
- ■ Choron 5
- ■ Joker Poker
- ■ Advanced Dungeons & Dragons
- ■ Final Assault
- ■ Evil Gorden
- ■ Mission: Con-Bat
- ■ Autoduel
- ■ Tower Toppler
- ■ TeleEpic & TeleWar II

PRODUCTIVITY:
- ■ Lattice C++
- ■ Publishing Partner Professional
- ■ Professional DataRetrieve
- ■ ComicSetter
- ■ Professional Draw
- ■ Draw 2000
- ■ Superbase Personal 2
- ■ Superplan
- ■ Deluxe Print II

UTILITIES:
- ■ B.A.D. (hot floppy and hard disk accelerotor)

ACCESSORIES:
- ■ CMI MIDI 1
- ■ INTRUDER ALERT

PLEASE CALL FOR PRICES

SOURCE LEVEL DEBUGGER 49.96
SPACE CUTTER 19.95
SPACE MATH 29.95
SPACE RACER 19.46
SPACE STATION OBLIVION 32.46
SPACEQUEST 33.00
SPELLBOUND 25.95
SPELLER BEE 31.25
SPELLING IMPROVEMENT 25.95
STAR WARS (EMPIRE) 30.25
STARGLIDER 30.95
● STARGLIDER II 29.95
STARRAY 23.61
STELLAR CONFLICT 26.95
STELLARYX 25.95
STOCK MARKET-THE GAME 16.95
STREET CAT 16.95
STREET SPORTS BSKBALL 13.95
STRIP POKER 27.46
STRIP POKER DATA #4 ..... 12.95
STRIP POKER DATA DISK #5 ..... 12.95
STRIP POKER II 25.95
STUDIO FONTS V1 (COLOR) 29.95
STUDIO MAGIC 64.95
SUB BATTLE 24.95
SUMMER CHALLENGE 9.95
SUPERBACK 51.95
SUPERBASE 49.95
SUPERBASE PERS. II 94.46
● SUPERBASE PRO. 3.0 ..... 229.95
SUPERSTAR ICE HOCKEY 29.86
SUPERSTAR ICE HOCKEY 32.46
SURGEON, THE 31.95
SWORDS OF SODAN 32.46
SYNTHIA 59.95
T SHELL 32.46

ZOETROPE 89.95
ZOOM 21.95
ZORK TRILOGY 47.95
ZUMA FONTS VOL 1-4 ..... 68.95
ZYNAPS 23.95

HARDWARE

68020-12 CPU/PROCESSOR 170.00
68030-16/20 CPU/PROCESSOR 570.00
68881 CRYSTAL OSSC. 16MHZ 5.00
68881-16 MATH COPROCESSOR 200.00
68881-20 MATH COPROCESSOR 300.00
68882-20 MATH COPROCESSOR 500.00
68882-25 MATH COPROCESSOR 620.00
AIR DRIVE 3.5" EXT DRIVE 169.00
AIR DRIVE 3.5" INT A2000 ..... 149.95
AMAS MIDI SAMPLER 109.95
AMIGEN GENLOCK 149.00
AMPLIFIED SPEAKERS 2x50 W 84.95
APRO DRAW 12x12 NO CURS. 464.95
APRO DRAW 12x12 W/CURSOR 499.95
BOING OPTICAL MOUSE 114.95
BYTE BOX 0K-RAM OPTIONAL 249.00
CA-880 FLOPPY DRIVE 159.00
CBM A2300 INT GENLCK 2000 ..... CALL
CBM A520 COMPOS. ADAP 500 ..... 49.95
CBM AMIGA MOUSE 65.00
● CBM DOS 1.3 ROM 39.95
CHEETAH MKS MIDI KEYBRD 235.00
CLTD TIMESAVER ..... 57.
CMI MIDI I (EXT) 52.6
COLOR SPLITTER 79.95
COMMODORE COLT PC 640K 599.00
CPS 500-POWER SUPPLY A500 ..... 74.95
CSA DRAGSTRIP RAM ACCEL. 723.45

FREE SHIPPING!*

CREATIVE COMPUTERS is both a mail order company with a store's support and three store showrooms with mail order prices. Visit one of our showrooms today. You'll be amazed!

3 RETAIL SHOWROOM LOCATIONS TO SERVE YOU

SANTA MONICA & WESTSIDE: 318 Wilshire Blvd., Santa Monica, CA 90401. Tues.-Sat. 11-7 p.m./Sun. 11-5 p.m./Phone: 213-394-7779

LAWNDALE & SOUTH BAY: 4453 Redondo Beach Blvd., Lawndale, CA 90260. Mon.-Sat. 11-7 p.m./Phone: 213-542-2292

VENTURA, OXNARD & SANTA BARBARA: 2112 E. Thompson Dr., Ventura, CA 93001. Tues.-Sat. 11-7 p.m./Sun. 12-5 p.m./Phone: 805-652-0325

COMMODORE AMIGA™

THE CREATIVE COMPUTERS ADVANTAGE:
- Authorized Amiga Dealer
- The largest dealer of Amiga products in the World
- Three store locations means excellent support
- Amiga-specific—unlike the competition, we don't claim being Amiga specific while selling other brands under another business name
- Authorized service center
- Uniform low pricing and largest selection, no hidden costs or catches
- We don't charge your card until the product ships

CUSTOMER SERVICE: Call 213-542-2292 from 11 a.m. to 7 p.m. PST for technical questions.

VISA · MasterCard · American Express Card · DISCOVER

***SHIPPING INFO:** No surcharge for Visa/MasterCard; 4% surcharge for American Express; 1% surcharge for Discover. Shipping within the continental United States for software orders over $100 (U.S.) is FREE. Please call for all other shipping information. Educational, corporate and aerospace purchase orders accepted.

RETURN POLICY: Defective merchandise under warranty will be repaired or replaced. Returned products must be in original condition and packaging. No refund on defective products or products that do not perform satisfactorily. We make no guarantees for product performance.

CONDITIONS: Creative Computers reserves the right to limit the sales of any items to local in-person pick-up only. Prices and availability subject to change without notice.

Creative Computers

CREATIVE COMPUTERS GOES INTERNATIONAL!

TOLL-FREE NUMBERS: Minimum orders $100 (U.S.). Orders only please. Absolutely no product inquiries or any questions will be taken on these lines.

INTERNATIONAL PHONE NUMBERS: Canada 1-800-548-2512; Italy 1673-74085; United Kingdom 0800-89-1178; France 19-0590-1099; W. Germany 01-30-810466; Japan 0031-11-1351; Australia 0014-800-125-712; Netherlands 06-022-8613; Norway 050-12029; Denmark 0434-0297; Switzerland 046-05-3420.

INTERNATIONAL ORDERING POLICIES: We have operators versed in the following languages: English, French, and Spanish, only. Shipping prices vary and are quoted following your order. Payment is made via credit card or electronic fund transfer. For any other information please call or regular toll lines. Absolutely no shipping inquiries will be accepted over the toll-free phone lines.

WE CAN BEAT ANY ADVERTISED PRICE! BUT WE SELDOM HAVE TO — OUR PRICES ARE THE LOWEST! (IF YOU SEE A LOWER PRICE ON AN ITEM, GIVE US A CALL.)

ACCESS

The world of non-commercial software

by Steve Ahlstrom
Copyright © 1989 Steven Ahlstrom

As the primary sysop on CompuServe's Amiga forums, and a veteran Amiga programmer, Steve Ahlstrom sees just about every notable public domain and shareware program that gets written, and in this column he tells you about the ones that have recently caught his eye. If you have some information you'd like to share through this column, you can leave a message to Steve on CompuServe (his PPN is 76703,2006), or send the info to us and we'll pass it on.

AmigaZoo 2.0 by J. Brian Waters

Programs like ARCHIVE, AmigaZoo, PAK - and I'm sure there will be others in the future - were written mostly for use in telecommunications. They make it easy to transfer a program via modem along with any associated files it may require. They allow you to create a single file that can contain many other files. The single file is normally significantly smaller, usually 50 to 60 per cent smaller, than the total size of the files contained within. This is achieved by various compression techniques. When transferring files to or from a commercial network or long distance BBS, the smaller the file, the less time connected and the less charges you incur.

These programs are also used to reduce the storage size of information you may want to keep available to you. They are handy indeed, allowing you to store twice as much information on a disk as you could without them.

Very shortly after the Amiga was released, Raymond Brand ported the ARCHIVE utility by SEA (System Enhancement Associates) to the Amiga. It became the de facto standard in librarian and file compression programs for the Amiga by default - at the time, there were no alternatives.

ARCHIVE (ARC) was perfect for Amiga's needs. It librariied and compressed files, and it was compatible with versions of ARC for other machines. You could download text files created on MS-DOS machines, STs, and anything else that had the utility available, and unARC them on the Amiga for your own use. Unfortunately, Amiga's ARC has not kept up with the advances in the program in the MS-DOS world. There are more and more times now that I'll download an ARC file created on an MS-DOS machine, only to be greeted, when I try to unARC on the Amiga, with the "You need a newer version of ARC" message. This is because newer compression techniques have been added to the MS-DOS version. The Amiga version has been dormant for a while.

Two other programs that provide the same functionality as ARC (plus more) are becoming popular: PAK and AmigaZoo (Zoo).

I'm not too crazy about PAK for reasons I listed in a previous issue - mostly poor compression. Zoo, on the other hand, is gaining in popularity for a variety of reasons.

AmigaZoo is compatible with the MS-DOS version. It appears to be well supported, maintained, and upgraded as the MS-DOS version is upgraded. Besides some of its technical advantages over ARC, the biggest advantage, for me, is that it *is* being actively supported.

Unlike ARC, Zoo supports full-length AmigaDOS filenames, and can archive and restore directory structures. This is a nice feature. Many times, using ARC, if you had a set of files with names longer than the MS-DOS standard allowed, or the program required some of its files to be in subdirectories, you would have to write a script (or use another program to create a script for you), and include that script in the archive so that those downloading the program could successfully set it up.

Zoo is friendlier to single disk drive users than ARC is. The single drive user can keep the Zoo program in his c: directory; when it is run it will wait for a carriage return before doing any disk I/O, allowing the user to change disks. He doesn't have to have Zoo, the .Zoo file, and the resulting destination file(s) on the same disk.

Zoo has many more features than ARC, yet those features you are used to with ARC have the same command line arguments. Little relearning is required. Figure 1 is the result of typing **Zoo H** on the command line.

Since AmigaZoo 2.0 has been released a few minor bugs have been found. These are being fixed shortly after they are reported. Recently a 2.01 patch file was released to fix a problem with the date stamp.

While Zoo is not yet as widely used as ARC, it is gaining popularity because of its features. Sometimes, depending on the file type, ARC is more efficient in compression on the same file. I'm willing to accept this. The difference is usually small and Zoo really shines by allowing full AmigaDOS pathnames and the archiving of subdirectories. Any loss in compression is usually more than made up for by having to supply a script with ARC files to recreate the correct names and directory structures.

I still use ARC because old habits are hard to break. But Zoo is slowly becoming my archive utility of choice.

AmigaZoo is a copyrighted shareware program. To quote from the documentation:

"AmigaZoo is a user supported program. Zoo was written by Rahul Dhesi and modified to run on the Amiga by J. Brian Waters. With the consent of the author, I am requesting a $10-$20 contribution from satisfied users. This contribution is voluntary and you need not feel any guilt if you do not contribute. Any contributions received will go toward one of three things: firstly, and most importantly, they will be used to acquire any tools which may appear on the market to help make AmigaZoo better, like more optimizing C compilers or better debuggers; secondly, it will be used to help defray the costs of distributing Zoo; thirdly, if any money is still remaining it will be used to buy some shares of Commodore stock."

AmigaZoo is available in CompuServe's AmigaTech library 4 as ZOO.ARC. AmigaZoo's author, J. Brian Waters, may be contacted by sending CompuServe email to 73127,2124.

```
Zoo archiver, Amiga Version 1.71 (J. Brian Waters 1987/12/28)
(C) Copyright 1986, 1987 Rahul Dhesi -- Noncommercial use permitted
Usage: zoo {acDelLPTuUvx}[acdEfInMNoOpPquzl:/.] archive file
       ("zoo h" for help)

Choose a command from within {} and zero or more modifiers from within []
E.g.:  'zoo a save /bin/*' will archive all files in /bin into save.zoo

 Commands in {} mean:          |Modifiers in [] mean:
  a     add files              | a     show archive name(s) in listing
  c     update comments        | c     add/list comments
  D     delete stored files    | d     extract/list deleted files too
  e,x   extract files          | dd    extract/list only deleted files
  l,v,L list filenames         | E     erase backup after packing
  P     pack archive           | f     fast add (no compression) or list
  T     fix archive datestamp  | M     move when adding (erase original)
  u     add only newer files   | n     add only files not already in archive
  U     undelete stored files  | N     send extracted data to Neverland
 ----------------------------- 0,oo  don't ask "Overwrite?"
  q     be quiet               p     pipe extracted data to standard output
  :     don't store dir names  /,//   extract full pathnames
  .     pack to current dir    I     add filenames read from stdin
  P     pack after adding      @n    start extract/list at position n

Novice usage:  zoo -cmd archive[.zoo] file...  where -cmd is one of these:
-add -extract -move -test -print -delete -list -update -freshen -comment
```

Figure 1: Zoo's help information

SetCPU 1.4 by Dave Haynie

Are you one the lucky people to have the A2620? If so, you need SetCPU!

SetCPU is intended to allow you to detect and modify various parameters related to 32 bit CPUs (68020 and 68030) in Amiga computers.

When run with no parameters the program will give you information about your system. Unfortunately, on my system, a run looks like this:

```
2> setcpu
SYSTEM: 68000
```

On a system with the A2620, a normal exchange would be:

```
2> setcpu
SYSTEM: 68020 68881 68851 FASTROM (INST: CACHE)
```

What this tells us is that you are running with a 68020 CPU, 68881 math co-processor, 68851 MMU, and have moved the ROM routines into 32-bit RAM. ROM routines in 32-bit RAM? Yes! That is what this program is really all about. You will see a dramatic speed improvement in all system calls by having the ROM located in 32-bit RAM. This feature isn't necessarily limited to the A2620. In theory it should work with third party boards with MMUs; however, it hasn't been tested by the author on boards other than the A2620. It will have to be modified a bit to work properly with memory outside of the 68000's 24-bit address space.

The syntax of the program is:

```
SetCPU  [INST|DATA] [[NO]CACHE|[NO]BURST]
   [[NO]FASTROM [TRAP]] [NOMMUTEST]
   [CHECK 680x0|68851|6888x|MMU|FPU]
```

What does all that mean? Well, it's worth looking at in detail (these descriptions are from Dave Haynie's documentation):

[NO]CACHE: This command will switch on or off 68020 and 68030 caches. If not qualified, it'll act on both instruction and data caches of the 68030.

[NO]BURST: This command will switch on or off the burst cache line fill request of the 68030. If not qualified, it'll act on both instruction and data caches.

INST: This precedes a CACHE or BURST operation to restrict its application to the instruction cache only.

DATA: This precedes a CACHE or BURST operation to restrict its application to the data cache only.

[NO]FASTROM [TRAP]: This activates the FASTROM translation on or off an MMU-equipped system. When switching on,

it first allocates 256K of memory for the ROM image, then 512 bytes of memory for the MMU table. It copies the ROM into the image area, then applies the translation by pointing the MMU at the table and activating it. The TRAP option causes the MMU to look at all 32 address bits; access to any memory outside of the 24-bit space will result in an exception which, if unhandled, results in a Guru 2. When TRAP is not specified, the MMU is only looking at 24 bits of address space. This is the mode you'd normally run in. Under V1.3 and earlier releases, a DOS bug can cause invalid accesses, which cause the exception, when running the EndCLI or NewCLI/NewShell programs; running untrapped will avoid gurus with these commands.

It's still possible to run into an exception when running without the TRAP option, however. One easy way is by writing to the ROM image, which is write protected by the MMU. The NOFASTROM option will switch off the MMU and reclaim the memory used for the ROM image and MMU table.

Note that this is achieved by locating these items via the appropriate MMU registers. If any other program set up the MMU for something, this could be a very bad thing to do. In general, until there's some level of operating system support for the MMU in Amiga systems, you're really safe using only one MMU tool at a time.

NOMMUTEST: This option won't normally be necessary. However, some 68020 boards out there apparently have hardware bugs of some sort that make testing for the MMU impossible, at least as near as I can tell. If your 68020 board freezes, or crashes with some kind of coprocessor protocol violation GURU (probably numbers $0D or $0E), you have a problem with your 68020 board. The most common bug is the 68020 board hardware failing to fully decode CPU space, such that the FPU in the system also responds to the MMU's address space.

CHECK: This option lets you check for the existence of a particular CPU system component in a script. It works like this:

```
SetCPU CHECK 68020

If WARN
   echo "No 68020 here!"
Else
   echo "Sho nuff got a 68020 here!"
Endif
```

The arguments to CHECK can be any of:

68000
68010
68020
68030
68851
68881
68882
FPU (Matches 68881 or 68882)
MMU (Matches 68851 or 68030)

SetCPU, C source and executable, has been placed in the public domain by Mr. Haynie.

It is available in CompuServe's AmigaTech library 14.

PathAss 1.0 by Anders Lindgren

Even with its slightly naughty-sounding name, PathAss is a nice contribution to the Amiga shareware library by Anders Lindgren of Sweden.

What PathAss does is allow you to assign devices across directories or even disks. How often have you wished you could give a command like:

```
ASSIGN Fonts: DF0:font AND DF1:FONTS
```

With PathAss you can! Here's an example of the kind of thing you can do:

```
PathAss DEVS: vd0:devs,df0:devs,df1:devs
```

The order of the directories is significant: they are searched from first to last. You can easily list all current assignments or delete any or all assignments.

Another feature, which might not be considered such a good thing, is that the assignments are not volume relative.

In other words, if you say **PathAss FONTS: RAM:,DF0:**, PathAss will look in RAM: then in the root directory of the disk in DF0:, whether that was the disk when you made the assignment or not. It looks at the device, not the volume name. This could be handy if you have many disks with fonts on them for which you would like to have easy access.

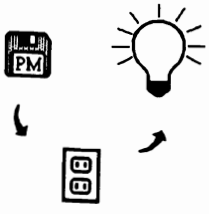
PathAss is shareware. The shareware fee is $10. The author will accept payment in "any currency, like swedish crowns, new PD programs, clean disks and so on".

PathAss is in CompuServe's AmigaTech library 14.

PowerMaster Amiga Home Control

Programmed Remote Control Of Lights & Appliances!

- Adds Convenience, Increases Security
- Uses X-10 Power-House System
- Amiga Not Required After Programming
- PowerMaster & Amiga Interface: $65 U.S. + $3 s/h ($6.50 Cdn.)



Compplications

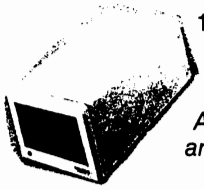
1727 Parkview
Redlands, CA • 92374
(714) 794-5311

DR. OXIDE SLICES PRICES!

Comp-U-Save's cost-cutting clinician is at it again! This month Dr. Oxide, a bargain-basement surgeon if ever there was one, offers the industry's lowest prices on hardware and software, plus special deals on products available only through Comp-U-Save!

BUILD YOUR OWN HARD DRIVE KITS!

5¼" powered hard drive chassis with fan
3½" powered hard drive chassis with fan
only $120.00 each



1000 or 500 SCSI interfaces with passthru and software for
$145.00

All you need is a SCSI hard drive and you're ready to Rock and Roll!

Call for SCSI hard drive prices

AMIGA PUBLIC DOMAIN

OVER 700 DISKS!

Largest Amiga PD Library in the World
also C-64 & C-128 – Write for Free Catalogue
Amiga PD – $4.00 each

EXTERNAL SLOT EXPANSION FOR AMIGA 500 and 1000

Now you can use most of the expansion cards designed for the 2000-hard disk controllers, 2/4/8 meg RAM cards, A2088 Bridgeboard, etc. Use low cost IBM compatible expansion cards already supporting a wide range of business and scientific applications. Designed to work with auto configurable cards. Meets Zorro Bus and Amiga 2000 Bus electrical specifications. Available exclusively through Comp-U-Save!

A500 2 SLOTS — $188.00
A1000 2 SLOTS — $188.00
A1000 3 SLOTS — $249.99

Amiga Hard Drives 500 - 1000 - 2000

20 Meg - $5.85.00 32 Meg - $699.99 48 Meg - $799.99

Amiga External Drive $139.99

Only Uses Half the Power of 1010 - with Pass thru

Panasonic WV1410 Video Camera

For Digitizers - $199.99 16 MM Lens - $29.99

Special 2400 Baud Modem - $149.99

★ Trackball	$45.00
★ Plastic Diskbank (Holds 120 3.5 in. Disks)	$14.99
★ Copy Arm (Heavy Duty)	$29.99
★ Mouse Mat (Fabric)	$ 5.00
★ Mouse Mat (Teflon)	$11.00
★ Gender Changers – All Types	Call
★ Static Mat (23.5 x 25.5 in.)	$24.00
★ Rapid Fire Joysticks	$12.00
★ Printer Buffer (32K–512K)	Call
★ RF Modulator (A1000)	$14.99
★ A/B Switch (Ser.)	$13.99
★ A/B Switch (Par.)	$14.99
★ A/B/C/D/ Switch	$29.99
★ Crossover Box	$39.99
★ Cables 500 – 1000 – 2000	Call
★ Teak Diskbank (holds 150 3.5 in. disks)	$39.99
★ Teak Diskbank (holds 200 5.25 in. disks)	$39.99
★ The Library (holds 80 3.5 in. disks)	$19.99
★ Floppy Wallets (Many Sizes)	Call
★ 3.5 in DS/DD Disks (Bulk)	$1.25 ea.
★ 5.25 in. DS/DD Disks (Bulk)	$ .39 ea.
★ Books — All Titles 20% Off	Call
★ Memory 512K — 8 Megs	Call

★ **We carry tons of hardware and software for the Amiga. Call for prices.**

Comp-U-Save

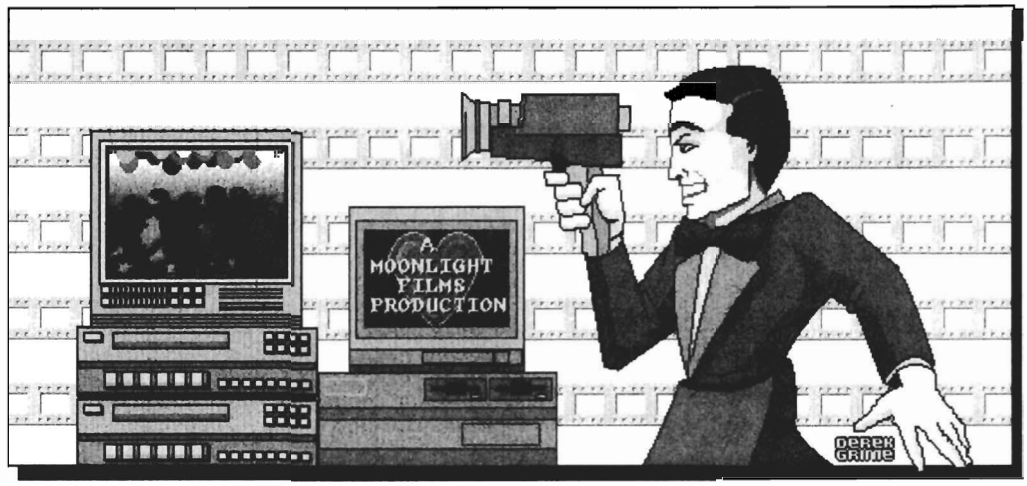
410 Maple Avenue, Westbury, NY 11590
In NY State (516) 997-6707 (Tech Support) Outside NY State (800) 356-9997 (Orders Only) Fax (516)334-3091

Come See Dr. Oxide
In Our Booth At All
World Of Commodore Shows!

Mastering The Raster

Turnkey on a dime

by Derek Grime



Amiga graphics are used in television production on a regular basis. Whether it's on MTV or a local real estate channel, barely a week goes by without a new sighting of our favourite machine strutting its stuff on the airwaves. We know it is possible to produce broadcast worthy computer animation with the Amiga. Getting the quality that producers need from the Amiga means pushing it to its very limits. The techniques are difficult to master. Deadlines are tight. Expensive third party hardware is often required. Realistically, very few Amiga users want to spend the time or money involved to achieve broadcastable results.

The typical director-to-be wants plain factual information spoken in simple English. There is a lot of confusion in the Amiga community regarding the best software and hardware for video production. This month let's focus on a typical user problem and discuss some of the solutions. We'll keep the jargon to a minimum. We promise to address the heavy issues facing the computer graphics professionals in the months to come.

For absolute beginners

Let's assume that you are a fledgling videophile. You own a couple of nifty consumer VCRs and a camera. Your artistic talents are minimal but your video abilities are abundant. At family reunions you wander from table to table clutching your camcorder, seeking that one candid moment that will haunt Uncle Malcolm for the rest of his life. At parties people actually request to see your home-brewed productions. In marketing parlance you are a prosumer, a consumer who demands the best results and has the disposable income to make it a reality.

Computer graphics fascinate you and you realize the old IBM in the guest bedroom just isn't going to cut it. You may or may not own an Amiga at the moment, but you've seen the demos and it looks like something you'd like to pursue. You're not sure where to start but you've got some interesting ideas you want to try out.

You feel that if you could take your home video camera and talk to some friends, you may have the chance to produce some wedding videos. Amiga graphics and titles will give you a sizable edge over the local competition. As a bonus, the extra money you earn will come in handy when it comes time to convince the rest of the family you have to have a 2500. As a computer animator one of the questions I hear most often is "what do I need to get started?". The simple answer is "not as much as you think".

Hardware

Let's begin with the hardware. If your ambitions are modest you will be able to function quite well with an Amiga 500. Although it is Commodore's entry level machine in the Amiga family, it can basically do everything that its bigger and more costly brother can. The resolution is the same high quality, there are just as many colours on screen, and it is just as fast as the 2000. It also costs less than half the price. The major drawback of the 500 is that it is not nearly as expandable as the 2000. While this is nothing to be concerned with at the moment, it will make a bigger difference in the years to come.

Right now incredible video cards are being developed for the 2000 and not all of them will be refitted to work on the 500. Compounding this is the fact that the stock Amiga 500 does not have a robust power supply. This means that your options are limited. An extra 3.5" drive, a printer and a modem won't tax the electrical system. After that you can safely add only one major extra peripheral to the computer's single expansion bus slot. Most owners either opt for a RAM expansion or a hard drive. A hard drive is a terrific time saver but for video graphics, two megabytes of RAM is almost a necessity.

The money you save now by buying a 500 can be used towards an extra VCR. When the new video hardware materializes, you can always sell your machine and trade up to a 2000. Of course if you are diving in head first, purchase a 2000 now and prepare for the future today. If all you are interested in is creating smooth three dimensional graphics with textures and multiple light sources, then go all out and get a 2500. Three dimensional rendering is one of the most difficult problems for a computer to deal with. The average Amiga can take days drawing a single frame of an animation with all 4096 colors. This top of the line Amiga has the faster 68020 processor to better handle the heavy math situations it will be facing. It will save weeks of your time in the long run.

Software for still images

Getting ideas into the Amiga is the first problem you'll face. Consumer level videos usually contain two types of computer graphics. The first is the spot illustration. This is a title card used to introduce the next section in your video. Television networks use spot illustrations before they break for a commercial. Hundreds are stored on a hard drive and called up when needed. The same ones are used again and again, sometimes with minor changes. A wedding video could contain a number of spot illustrations. There could be one used before the preparations, the ceremony, the receiving line, the reception and the hasty exit of the bride and groom. In the past, these were always prepared by hand and then shot on a copy stand. Most small video houses still do it this way even though it's the same technique that has been used since the silent movie era. Traditional title cards are static and boring and are difficult or impossible to edit. With the Amiga, you can keep a file of common images on diskette. They can be loaded in and altered at any time, giving every new production a personal touch. Colour cycles and simple animated effects can be added in minutes. After you get a few jobs under your belt, a clip art collection can handle almost any situation. Even if you have trouble drawing stick figures, excellent compilations of graphics are available at reasonable prices.

Excellent paint software abounds for the Amiga. The best of the lot is Electronic Arts' Deluxe Paint II. Deluxe Paint was introduced almost three years ago and has aged gracefully. It lacks some of the features of the newer packages but none of the competition can even come close for ease of use. In your new business this is the most valued quality a paint program can have. The user interface is simple and efficient, often copied but never duplicated. Deluxe Paint works in all resolutions, allowing from two to thirty-two colours out of a total palette of four thousand and ninety-six. Thirty-two colors may seem limiting, but how often do people colour a picture using the whole deluxe box of crayons?

All the tools that you would expect in a high-end paint program are here as well as some you wouldn't. Symmetry radiates your drawing outward from a central point in real time. Objects can be filled in a gradient range of colours, giving startling effects. Stencils can be used to lock colours in your picture so that you can actually draw behind them without affecting the original drawing. Finally there is colour cycling. Colour cycling steps through a swatch of preselected hues at any speed. Cycling can be used for simple animation effects like streaking comets, fireworks and star fields. Cycles can make a drawing pulse as if it's radioactive. Cycling an entire palette will make the artwork appear to transform itself inside-out. If you buy only one piece of software for the job, it should be Deluxe Paint II. A promised upgrade that will be available soon even has built-in full screen animation.

The newer entries in the paint play-offs offer the ability to have all the Amiga's colours on the screen at once. The overall effect is much smoother and looks richer and warmer. The three that are most preferred are NewTek's Digi-Paint, MicroIllusion's Photon Paint and Electronic Arts' Deluxe Photolab. There is no clear winner. Photolab has the most features and comes with some great extra software. Photon has a friendly user interface, similar to DPaint's. Digi-Paint handles colours and tinting in better ways, but lacks many of the features of the other two. If you want to work with these more powerful graphics modes, find a friendly dealer and see if he'll let you take a test drive. If it feels good, buy it.

Rolling the credits

Titles and credits are an integral part of any video that you produce. So far there is no one piece of software that is both powerful and simple to use. If moving words are important, take a look at JDK Images' Pro Video CGI. The name doesn't exactly roll off the tongue and the user interface is quite different from most other Amiga offerings. The price is high compared to the rest of the pack, but you get what you pay for. Pro Video CGI is truly a professional level product. In fact the smooth hi-res titles it creates are used widely in the broadcast realm. Patterned after costly character generating computers, this titler rules its own niche market and is responsible for getting Amigas into many post-production houses.

Taking a different course is Aegis' VideoTitler. The included fonts can be resized in any direction without losing clarity. Fonts and pictures can be pushed back at all angles of perspective. A keystroke will change the words on the screen to different styles including flashy effects like neon, embossed and extra wide drop shadows. VideoTitler will show graphics in an overscan screen as large as 768 by 480. This is as high as the Amiga can go, and VideoTitler is one of the very few pro-

grams to support it. As well as the text editing you also get a slide show maker for your money.

Broadcast quality?

Getting the graphics out to videotape is the most important part of the job. Advertisements in Amiga publications are fond of using the enigmatic term "broadcast quality". While there is such a thing, many vendors have obscured the meaning beyond all hope of recovery. Ignoring the specifics, you should only be after broadcast quality if the VCR you are going to be using in your work costs over fifteen thousand dollars. Frantic consumers everywhere demand broadcast quality output from their Amiga 500s, not really taking the time to think about their needs.

Let's clear away the hype. Hook an eight hundred dollar Amiga to each of the genlock/encoders available. Run the video into a quarter million dollar suite as we have done. Out of all the various products we've tested this way only a couple live up to the bill. They all retail in the two to three thousand dollar range. My loose definition of quality is anything a video engineer doesn't start complaining about. Video specs are their life so making them happy is a helluva task. Any edit suite worth the amount you pay to use it can take whatever signal you throw at them and turn it into something beautiful. They'll pull it out of the RGB port if they have to. We've gone to one inch tape with everything from the Amiga 1000's 1300 genlock right up to the 2000's Magni. We've had networks screen our work from VHS, though I imagine the engineers swore a blue streak. Fact is, broadcast quality is what you want it to be.

In the real world you are probably editing on VHS tape or some other consumer format. A three thousand dollar genlock will look fantastic on VHS. So will a three hundred dollar genlock if you take care. You see, despite their many faults, home video decks are very forgiving. They take what they get and do their best. This past week our studio received a call from an outraged Amiga 500 owner. He paid two thousand five hundred dollars (Canuck bucks) for his genlock only to be very upset with his transfers. The problem was the colours flared and bled as if they were alive. Delving further into the problem, we learned that the red spectrum was the worst offender. Not surprisingly, he was transferring to a four hundred dollar VHS machine. Let's get this on the record. No genlock, no matter how much it lightens the wallet, is going to get VHS to handle one hundred percent red. It just can't do it. You see, the television standard that we use in North America wasn't even designed to show colour in the first place, let alone the hot ones. VHS is great for all of us couch potatoes on Saturday nights, but it does have its shortcomings. Red is only one of many. Our Amiga is a graphics computer. It will give you a super saturated hue if you want it. But don't expect to record it. Unfortunately, the caller was less than convinced by our suggestions and was going straight to the manufacturer. You can imagine the scene when he first walked into the computer store waving his Amex card. In sales talk this translates as "a fool and his money are soon spotted".

The cat is coming out of the bag, folks. All of you working out of your homes on home equipment can do quite well with a cheaper genlock. The big ticket work prefers the big ticket genlocks, but the inexpensive ones will do the trick until things get rolling. Commodore's own genlock, the 2300, is one of many that will fit the bill. There are also some dogs running with the pack, so as always attempt to try before you buy.

Video secrets of the Amiga gods

Here's the payoff for bearing with me this month. Scanning over six feet of various Amiga publications on the bookshelf has yielded some surprises. These simple rules of good Amiga video production have never been presented all at once anywhere else. Some have never been explained anywhere.

Watch the colours: Always keep the colour values to less than twelve. The Amiga ramps right up to level fifteen and that's just too much for any format of video. Reds are the worst, followed by purples and yellows. Blues transfer the best. Check the palette before you go to tape.

Interlace: Always have it on when you dump to tape. The better genlocks do it internally but you can run a public domain program like SetLace in the background that does relatively the same thing. Video is interlaced and giving it an interlaced feed from the Amiga makes things much cleaner. Of course there is nothing to worry about when working in video/ interlace or high resolution.

Overscan: Make sure the software you buy supports it. The more extreme the better. The real world has no borders. On second thought, make that the televised world.

Preferences: Make some adjustments beforehand to make sure the picture is centred from the genlock. It may look fine on the monitor but the VCR sees it differently. Many jobs are ruined because the preferences weren't checked before the transfer.

Big is better: At least in computer graphics. A VHS deck has less than 250 lines of resolution. In high-res mode you can go to nearly triple that. Single pixels will get lost and one-line borders are a waste of time. Heavy text always looks better, and the television you show the work on is not nearly as good as your 1080/4 monitor.

Moonlighting

Desktop video is gaining momentum and it will be the hot computer topic in the nineties. It all began with the Amiga, and nothing does it better.

Computer graphics on video is an exciting field to be involved in. Your Amiga is one of the few hobbies that can satisfy both your mind and your wallet. It's also a great way to afford all the new products that come along every month. The way things are exploding, it may be the only way.

InterComputing Inc.

2100 N. Hwy 360, Suite 2101
Grand Prairie, TX 75050-1015

Orders only call : 1-800-622-9177 (now incl. Texas)

Customer Service and Order Status please use
1-214-988-3500

Amiga is a trademark of Commodore-Amiga, Inc.

AMIGA

We are in business since 1984;
We support the AMIGA since 1985;
We will give you the best service possible at prices the other 'Discount-places' wont like - PERIOD !
Please call for a complete listing our products !
Member Better Business Bureau in Dallas

HARDWARE

Memory

8-Up Memorycard 0K	145.75
8-Up Memory with 2MB	649.75
8-Up Memory with 8MB	22.95.00
Starboard 0 MB A1000	239.75
Starboard 0.5MB A1000	399.75
Starboard 1 MB A1000	599.75
Starboard 0 MB A-500	239.75
Starboard 0.5MB A-500	399.75
Starboard Upper Deck 0k	44.75
A1000 256k Front Panels	199.00
A501 - 512k w/clock	call

Drives

FData-10 ext. Drive	139.75
FData-20 ext. TwinDrive	249.75
int. 3.5" Drive A2000 only	145.75
AIR-Drive	169.75
Cal.Access CA-880	169.75
HardFrame SCSI Card	239.75
Starboard SCSI Module	97.75
Quantum 80MB Drive	875.00
Supra 20MB HDisk	649.00
Supra 30MB HDisk	749.00

Don't Pay More !

Flexible Data Systems

FDATA-10 Disk Drive

This external 3.5" Drive is a very inexpensive unit to make the life with your Amiga easier.
The FData-10 is compatible with the original Amiga-drive but is quieter, smaller and has a longer cable (27inch) to connect it to the Amiga.

Oh yes, before we forget to tell you,
it's only **$139.75 !**

Video / Graphics

A-Pro Draw 12x12	499.75
A-Pro Draw 9x6	399.75
Amiga Live A 500	299.75
Amiga Live A 2000	329.75
Amiga Live A 1000	219.75
Color Splitter	85.75
Digi View 3.0 NTSC	139.75
Digi View Adapter	19.75
Digi Droid	64.50
Easyl 500	299.75
Easyl 1000	349.00
Easyl 2000	349.00
Flicker Fixer	499.00
Flicker Master	12.50
Gen One Genlock	749.75
Imprint	399.75
Mitsubishi Multisyc Monitors for use with FlickerFixer :	
13" Screen size	549.75
16" Screen size	1349.75
20" Screen size	1895.75
Panasonic 1410 Kit	249.75
Perfect Vision	175.00
Polaroid Palette	1995.00
Super Gen	699.75
X-Specs 3D	97.75

Software Deal of the Month

X-CAD Designer

199.75

if you Trade in any commercial CAD-Package (no PD-Stuff please) for any Computer !
This offer is valid until April 30th or while supply lasts !

Sound / Midi

ECE-Midi w/ RS-232 thru	44.75
Future Sound Sampler	129.75
Soundscape Sampler	79.50
Soundscape Midi Interface	42.75
Perfect Sound A500/2000	59.75
Perfect Sound A1000	59.75
Midi Cable 5ft	8.75
Midi Cable 8ft color coded	11.95
Midi Gold for A2000 internal	69.95

Miscellaneous

Epyx 500XJ	17.50
Mouse Pad	6.95
Mouse Time for A1000	29.75
Modem Cable A500/2000	9.95
Multifunction Card	69.75
Phasar Gun	46.75
Processor Accelerator	149.50
Sony 3.5" DS/DD Disks	18.75
Supra Modem 2400 Bd.	149.75

GVP-Products

20MB A500	639.75
80MB A2000	1149.75
Controller w/2MB	699.75

New in Stock

Heroes of the Lance	29.75
M	154.75
Sound Quest K1-Lib.	99.75
Page Stream	134.75
Lattice C 5.0	219.75
Professional Draw	134.75
Desktop Budget	44.75
B.A.D.	call

MUSIC

Audio Master II	69.75
Deluxe Music	64.75
Dr. T's 4-OP Deluxe	99.75
Dr. T's DX-Heaven	99.75
Dr. T's Kawai K1 Libr.	99.75
Dr. T's KCS 1.6	166.75
Drum Studio	29.75
Dynamic Drums	47.75
Dynamic Studio	129.75
Laser Sound Vol. 1-3	27.50
M	154.75
Midi Recording Studio	44.95
Music Mouse	49.75
Sonix	47.75
Sound Oasis	66.75
Sound Quest Casio CZ	84.75
Sound Quest D-110	99.75
Sound Quest D-50	99.75
Sound Quest DX-7	99.75
Sound Quest DX-7 II	117.75
Sound Quest Generic	84.75
Sound Quest MT-32	99.75
Sound Quest Q-80-ESQ1	99.75
Sound Quest TX-81 C	84.75
Soundsc. Pro Midi Studio	124.75
Soundscape Utilities I	39.95
Studio Magic	66.75
Synthia	66.75
Texture	99.75

Trade up!

We will trade in your Amiga 500, 1000 towards a new Amiga 2000 or 2500! Please call for details.

How are you, Jerry ?

BUSINESS

Analyze 2.0	62.75
BEST Management System	299.75
Critics Choice	159.75
Cygnus ED	69.50
Excellence	189.75
Financial Plus	199.75
Flow	59.95
Gold Spell	29.75
Kindwords	66.75
Logistix	99.75
Maxi Plan 500	99.75
Maxi Plan Plus	129.75
Microfiche Filer	69.75
Microfiche Filer II	112.75
Money Mentor 2.0	59.75
Organize	66.75
Phasar 3.0	66.75
ProScript	32.95
Prowrite 2.0	72.50
Scribble 2.0	57.95
Softwood File II (GS)	79.75
Superbase Personal II	99.75
Superbase Professional	199.75
TexEd	22.75
TexEd Plus	54.75
Word Perfect Library	79.95
Wordperfect	197.75

GRAPHICS / VIDEO

3-Demon	69.75
Aegis Animator	82.50
Aegis Draw 2000	179.75
Analytic Art	39.75
Animate 3D	99.75
Animation Multiplane	56.75
Animation Apprentice	199.75
Butcher 2.0	21.75
Calligrapher	87.75
Comic Setter	66.75
Deluxe Paint 2.0	94.75
Deluxe Photo Lab	99.75
Deluxe Productions	144.75
Deluxe Video 1.2	94.75
Director	44.95
Dynamic Cad	299.00
Fantavision	39.95
Flipside	39.75
Forms in Flight II	77.75
Gallery 3D	43.75
Interchange	29.75
Intro CAD 2.0	49.75
Kara Fonts	52.75
Lights, Camera, Action	52.75
Modeler 3D	72.50
Pageflipper Plus F/X	99.75
Photon Paint	66.75
Pixmate	49.75
Prism Plus	44.75
Pro Board	379.75
Pro Net	379.75
Pro Video Plus	199.75
Sculpt 3D	69.75
Sculpt 3D (europ.Ver.)	69.75
Sculpt 4D	399.75
Silver	99.75
Terrain Disk	19.75
TV Show NTSC/PAL	62.95
TV-Text	62.95
Turbo Print	32.75
Turbo Silver	129.75
Video Effects 3D	129.75
Video Titler	99.75
Videoscape 3D	119.75
Window Print II	22.75
X-CAD	349.75
Zoetrope	99.75

Win

a Yamaha Synthesizer complete with MIDI Interface and Cables !

All Orders placed between January 1st 1989 and May 31st 1989 are automatically entered in our drawing. There is no purchase necessary; If you don't need anything for your Amiga just send a postcard with your Name,address,Phone and Typ of Amiga (important, if you win you want the correct interface . . .) to :
InterComputing Inc.
Yamaha-Drawing
2100 N. Hwy. 360, #2101
Grand Prairie, TX 75050

DESKTOP PUBLISHING

Comic Setter	66.75
Font Set I	24.75
Laserscript	29.75
Newsletter Fonts	29.75
PCM-Fonts Vol.1-3	27.50
Page Setter	89.75
Professional Page	239.75
Shakespeare	149.75
Studio Fonts	29.75
Zuma Fonts 1,2,3,4 each	24.75

UTILITIES

Arexx	32.75
Disk Mechanic	56.75
DOS-2-DOS	42.75
Demonstrator	24.75
EZ-Backup	32.75
Facc II	19.75
Fine-Print	32.75
Project D	32.75
Quarterback	44.95
Raw Copy	39.95

PROGRAMMING - LANGUAGES

A/C Basic Compiler	129.75
A/C Fortran (AB-Soft)	179.75
Aztec C Developer	199.75
Aztec C Professional	145.75
Aztec Source Level Debug.	59.75
Benchmark Amiga Library	64.75
Benchmark C-Library	64.75
Benchmark IFF Library	64.75
Benchmark Modula-2	134.75
DSM Disassembler	42.75
Lattice 5.0	219.75
Lattice-C ++	395.00
Lint	69.00
MCC-Pascal	79.95
Modula-II Commercial	175.00
Modula-II Developer	89.75
Modula-II Standard	59.75
Power Windows 2.5	56.75

Hardware of the month.

X-SPECS 3D

Get a free Art Disk with every Set of X-Spects-3D if you order before May 30th 1989.

$97.50

GAMES

4th and Inches	29.75
Andromeda Mission	27.75
Amegas	19.75
Arazok's Tomb	24.75
Arkanoid	19.75
Balance of Power	33.75
Barbarian	24.75
Battle Chess	31.75
Black Shadow	25.75
Capone	29.75
Carrier Command	29.95
Centerfold Squares	22.50
Championship Basketball	29.75
Championship Baseball	27.75
Championship Football	29.75
Championship Golf	27.75
Corruption	27.95
Cube Master	19.75
Defender of the Crown	35.75
Destroyer	27.75
Ebonstar	29.75
European Scenery Disk	19.75
Faery Tale Adventure	29.75
Ferrari	32.75
Final Assault	31.75
Fire and Forget	28.75
Flightsimulator	39.75
Foot Man	22.75
Foundation Waste	27.75
Gee Bee Air Rally	27.75
Global Commander	29.95
Grand Slam (Tennis)	29.75
Guild of Thieves	29.75
Heroes of the Lance	29.75
Hollywood Poker	24.75
Hybris	27.75
Interceptor	34.75
Into the Eagles Nest	24.75
Jet	32.75
Jewels of Darkness	9.75
Karate Kid II	27.75
Knight Orc	29.75
Major Motion	27.75
Menace	21.95
Mindwalker	27.50
Obliterator	29.75
Offshore Warrior	28.75
Pioneer Plague	27.95
Plutos	19.75
Portal	34.75
Ports of Call	29.75
POW	25.75
Q-Ball	19.75
Realm of the Warlock	24.75
Return to Atlantis	34.75
Road Raider	27.75
Roger Rabbit	29.95
Scenery Disk 7/11	19.75
Scenery Disk Europe	19.75
Scenery Disk Japan	19.75
Shadowgate	29.75
Shanghai	27.75
Silent Service	27.75
Silicon Dreams	9.97
Sindbad	33.75
Skychase	27.75
Space Cutter	19.95
Space Quest	32.75
Starglider	29.75
Starglider II	29.75
Starwars	27.75
Street Cat	17.75
Street Sports	14.95
Strip Poker	27.75
Strip Poker Data 4 or 5	14.95
Surgeon	29.75
Terror Pods	29.75
Test Drive	29.75
The 3 Stooges	34.75
The Pawn	29.75
Torch 2081	19.75
Uninvited	33.75
Vampires Empires	29.75
Virus	19.75
Vyper	22.75
Wintergames	14.95
World Class Leaderboard	29.95
World Games	27.75
Zoom	21.75

And as allways;
we've got the most 'customer friendly' Terms :
S/H $3.95
30$ Minimum Order
NO Credit Card Fee !
but
in TEXAS add 7% TAX !
All prices are subject to change without notice

$3.95 S/H to Cont. USA only!
We ship to APO addresses.
RMA# required on all returns.
Restocking charge is 18%

Assembly Programming, Part 2

Twisting ECHO's tail

by Jim Butterfield
Copyright © 1989 Jim Butterfield

Jim Butterfield is a Toronto programmer, author and lecturer whose many articles, books, lectures and television programs have made his name a household word among Commodore users throughout the world since the early days of the KIM-1 microcomputer.

Looking back

We covered a lot of ground - and used a lot of words - last month in part one. This time I'd like to touch a few highlights, and add a little perspective.

In part one, we had to gallop hard to get to the point where a working program could be written. Exploring is fun, but it's time to read through this list, ensuring that our information base is well in place.

1. 68000 *data and address registers* can each hold 4 *bytes* of information. If you like, you may call this 32 bits, 2 words, or 1 long word. That means you can hold some fairly big numbers in there, and do arithmetic on them.

2. Data registers may be used in any of three ways: as the full *long word*, all four bytes; as a *word*, the two lower bytes; or as a *byte*, the lowest byte of the register. The part of the register you don't use will not be affected.

Address register operations change the whole *long word*. This is true even if you specify **.W** (word) mode; the two bytes you move will expand to fill the whole address register. And you can't use **.B** (byte) mode with an address register.

3. Address register 7 is the stack; don't change it without reason. Address register 6 is used in the Amiga for library calls; keep it fairly free for this use.

Address registers 0 to 5 and all data registers are free for you to use. You'll tend to use the higher numbered registers; that's because subroutine calls can lose the information in lower numbered registers, as the next item notes.

4. System subroutines (library calls) are careful not to change the contents of address and data registers 2 to 7; values you have there are safe. But registers A0, A1, D0 and D1 are likely to be changed by these subroutines; you'll lose any information you were holding in these.

5. The 68000 can reach memory either: one byte at a time (any address); or two bytes (a *word*) at a time (high byte at an even address, low byte at the next higher odd address). An instruction can ask to handle four bytes (a *long word*); in this case there will be two dips into memory. Thus, you could ask the processor to give you the long word contents of address 4, and it would fetch the contents of bytes 4, 5, 6 and 7, placing the result into the register you specified. If you tried to do a long word (or word) operation on address 5, however, the computer would refuse, and you'd meet the guru.

6. We have met a few 68000 instructions. For moving data, we have MOVE, which will move information (byte, word, or long word). LEA (Load Effective Address) moves an address (as opposed to moving data) into an address register.

These instructions have two *operands*: a 'from' and 'to', or, more formally, source and destination. MOVE usually specifies a data length: **.L** for long word, **.W** for word, or **.B** for byte. Thus, **MOVE.L 4,A6** moves the four bytes from addresses 4, 5, 6, and 7 into address register 6. LEA always moves a full address, which is a long word.

BSR (Branch subroutine) calls a subroutine that is sure to be close by (within about 32,000 bytes). If the subroutine were really close, within about 120 bytes, we may save a little memory by specifying BSR.S (branch subroutine, short). In contrast, JSR (Jump Subroutine) is able to call a subroutine anywhere in memory, perhaps one that is too far away to be reached by BSR.

BSR and JSR have one operand: the location of the subroutine you wish to call.

RTS returns from a subroutine to the program that called it. The entire program that you write is a subroutine, called by the system. When it's finished, it should return control to the system with RTS. RTS (Return from Subroutine) has no operands; it's complete in itself.

7. We have met a few *address modes*. Let's summarize the most visible ones.

Register direct means the contents of the register itself. Thus, **MOVE.L A0,D0** means "move the contents of register A0 into register D0".

Address register indirect means that the address register contains the address of the data. Parentheses are used to indicate an indirect address; only address registers may be used this way. Thus, **MOVE.L (A1),D0** means "A1 contains an address; move the contents of that address into register D0". In this example, the address in A1 must be an even number or the program is in trouble.

Address register indirect, with displacement asks for an indirect address again, but does a little extra arithmetic, offsetting (or 'displacing') the address by a fixed amount. Thus, **JSR -$228(A6)** means "get the address from A6, subtract hex 228, and do a subroutine call to whatever address results".

Absolute specifies a memory address in the Amiga. We rarely use fixed addresses, except for address 4, which contains a pointer to our Exec library. Even the input/output and system chips of the Amiga should not normally be referenced directly by the programmer; instead, a library call lets the system go after this data for you.

MOVE.L 4,A6 calls for the contents of address 4 (and 5, 6, and 7) to be moved into register A6. Note that this is different from **LEA 4,A6**, which would move 4, the address, into A6.

Amazingly, we may use absolute addresses to refer to something within our own program. At first, this is puzzling, since we don't know where our program will be when it's loaded into memory. However, everything works; the assembler and Amiga's loader work together to adjust the address so that it's correct within your program when it runs.

Thus, if we have a byte of information stored at address VALUE within our program, we might code **MOVE.B VALUE,D5** to transfer the contents of VALUE into the low byte of register D5. Whenever we can, however, we avoid this kind of absolute address in favor of PC addressing, to be described next.

Program Counter Indirect With Displacement is often called 'relative' addressing. It tells the processor to look ahead or back from its current working point (up to about 32,000 locations either way) to find the location of interest. If you are referring to data within the current block of your own program, you will often use this addressing mode.

You can't always use this addressing mode, but when you can it saves memory and computer overhead. Since the offset can be described in two bytes, and a full address would take a long word of four bytes, you save space and time. And this kind of address is valid no matter where your program happens to be loaded. In contrast, an absolute address referring to a location within your program would need to be adjusted according to where you program ended up.

Our previous example of **MOVE.B VALUE,D5** could be usefully changed to read **MOVE.B VALUE(PC),D5**. The resulting code would be shorter, and would not need relocation adjustment as it loads.

The (PC) mode, however, may be used only as part of a source operand, that is, only as the first address. If our example wanted to move the other way, we'd have to write **MOVE.B D5,VALUE** since (PC) would not be available.

Immediate addressing supplies an actual value to be used. It is signalled by the use of the # character (sometimes called the 'octothorpe' or 'pounds' symbol). Thus, **MOVE.L #0,A4** would put the actual value zero into register A4.

As with (PC) type addressing, immediate addressing may be used only as part of the first operand. So if I want to compare the contents of D4 with a value five, I must write **CMP.L #5,D4**, not the other way around.

There are other addressing modes... but we'll introduce them as they arrive.

8. There's a need for *startup code*, a set of instructions that do necessary overhead work. This includes such things as opening system libraries before the main job starts, and closing them after it is finished. Perhaps it would be more accurate to call it startup-and-closedown code.

The startup code we wrote last time assumed three things. First, it assumed that the program would be called only from CLI. That saved us quite a complex series of operations. Among other things, we could be confident that there would be a window to which to which we could send output. Secondly, it assumed that the only library we would need would be DOS. For beginners, that's handy; DOS is the simplest output path to screen, disk, printers or whatever. DOS is not much good for drawing pictures or playing music; but there will be time for that later.

Our third assumption - one which we will change today - is that the CLI command consists of the program name only. That is, the program would be invoked by typing **TEST**. If we should choose to type **TEST THIS PROGRAM**, the remainder of the line, **THIS PROGRAM**, would be ignored. We'll start to look at the question of reading the 'command tail', as it's called, in this session.

9. An observation: in writing a program that is completely self-contained, we've done something that's quite unusual in Amiga-land. Most programs call in extra data definitions from 'include' files as they start. Many programs use 'macros' to generate 'canned' code that doesn't need to be repeated. It is common for Amiga programs to call in pre-written startup code. We'll do that later, after we have written more complete

startup sections. Many programmers write and assemble complete sections of code; later, the linker will combine this code with other segments to make a complete operating program. Finally, it's almost universal to ask the linker to look up the library addresses for you; it's too much work to remember that -$228 is the OpenLibrary entry.

We'll open up this area only slightly at the moment. There are different rules that apply to different assembler/linker systems. We'll control as much as we can, so that there's minimal dependence on any special system features.

Next project: reverse ECHO

We know that CLI command ECHO repeats whatever it finds in the command tail; at least, that's its main job. Our project will be to write an ECHO command that repeats the information backwards. Not very useful, but we'll learn a couple of things on the way. We will call this program OHCE, which is ECHO spelt backwards.

Open up a new file called OHCE.ASM on your favorite editor, and let's get to work. A few comment lines, and then we'll do our system library definitions. They will be the same as in the earlier program; but this time, we'll let the system figure out their values.

```
; Reverse ECHO program
; Startup sequence, CLI only
; Definitions for Exec Library

 XREF _LVOOpenLibrary
 XREF _LVOCloseLibrary

; LVO means Library Vector Offset
```

XREF tells the assembler: "Leave a blank any time you see these labels, together with a note to the linker that this value needs to be filled in". The assembler will then go ahead with the job; even though it doesn't know the value of, say, _LVOOpenLibrary, it won't signal an error.

The linker, in turn, will read the object file and understand that it will need to find these labels in some other file: another object file or a 'library file'. When it finds the labels, it will fill in the correct values. Result: we end up with a complete executable program.

You can't use XREF with the AssemPro assembler, which doesn't do linking. Instead, substitute the command:

```
INCLUDE INCLUDES/EXEC.OFFSETS
```

This will automatically insert a set of EQU definitions for _LVOOpenLibrary and _LVOCloseLibrary, plus many others we won't need for this program. We'll do this again when we come to the DOS library definitions.

Catching the tail

Our previous startup code began by opening the DOS library. We have something new to do before tackling that. Information on the 'command tail' is given to us by the Amiga loader: A0 contains the address of this text string, and D0 contains its length.

Registers A0 and D0 will have their values disturbed when we call a system subroutine. So we'd better park their values somewhere safe. Any register numbered 2 or higher will do the job, so we'll code:

```
MOVE.L A0,A5
MOVE.L D0,D5
```

We could have picked any other convenient registers for these values, keeping in mind that A6 and A7 have other functions to perform. Now we may safely open the DOS library.

```
   MOVE.L $4,A6              ; Exec base
   LEA    DosName(PC),A1     ; Dos name string pointer
   MOVEQ  #0,D0              ; any version
   JSR    _LVOOpenLibrary(A6)
   MOVE.L D0,A6              ; Dos base pointer
   BEQ.S  Exit               ; Zero, quit
   BSR.S  Main               ; Do the job
   MOVE.L A6,A1              ; Dos base pointer
   MOVE.L $4,A6              ; Exec base pointer
   JSR    _LVOCloseLibrary(A6)
Exit:
   RTS
```

That completes our startup code. It's about the same as we have used previously, except for the XREF and the saving of our command tail registers.

On to the main job. As before, we'll make the library definitions into XREF statements, and then call to get the output stream file handle.

```
Main:
; Main Job
; Definitions for Dos library

   XREF _LVOOutput
   XREF _LVOWrite
```

Again, you can't use XREF with AssemPro. Substitute:

```
INCLUDE INCLUDES/DOS.OFFSETS
```

This will insert EQU definitions for _LVOOutput and _LVOWrite, plus many others. These .OFFSET files are in text form; you can read them if you like.

```
   JSR    _LVOOutput(A6)     ; Get output handle
   MOVE.L D0,D4              ; Handle to D4
```

We'll need to use our handle more than once... in D4 it will be safe from subroutine erasure.

Remember our command tail? Our job is to print it out backwards. We have the location of these characters in address register 5, and the number of characters in data register 5. The characters are bytes, of course.

We would use *Write* to print our character, of course. There's no problem printing the characters one at a time, and that's what we'll do, using a loop. More sophisticated programs would copy the reversed message into a work area of memory, and the write the whole thing at one shot.

By the way: the command tail string is *not* terminated by a zero byte. You'll usually find a newline (linefeed) character there. If you're testing for end-of-tail, it's usually safer to look for any non-printing character (any value less than 32 decimal). But we don't need to do that, since we know the length of the string: it's stored in D5.

I think we can safely assume that the string is less that 65,535 characters long. That means that we won't need to handle D5 as a long word; a word operation will get the complete value and save us a little (negligible) amount of run time.

```
Loop:
   SUBQ.W #1,D5
   BMI.S  Finish
```

The SUB (subtract) instruction will subtract 1 from D5. In this case, a small value is involved, permitting us to use the SUBQ (subtract quick) command. If the result goes below zero we will Branch Minus (BMI) to the loop exit. Otherwise, we have something to print:

```
   LEA    0(A5,D5.W),A0      ; Address of Character
```

Aha! A new addressing mode! This one is called "Address register indirect with index and 8-bit displacement".

That's quite a mouthful, but you may be able to see what it does by just looking at its appearance. The address in A5 is added to the counter value (word size) in D5; then zero is added to the whole shebang. On the first time around the loop, that will give the address of the *last* character of the command tail. As we go around the loop, decreasing D5, we'll back to previous characters. Of course, Write needs that address placed into register D2, so we take an extra step.

```
   MOVE.L A0,D2              ; ... to D2
   MOVEQ  #1,D3              ; Length of Character
   MOVE.L D4,D1              ; Handle
   JSR    _LVOWrite(A6)      ; Send Character
   BRA.S  Loop
```

To be neat, we must print a NewLine before the program terminates. Can we assume that register D3 still contains value 1?

```
Finish:
   LEA    NewLine,A0
   MOVE.L A0,D2
   MOVE.L D4,D1
   JSR    _LVOWrite(A6)
   RTS

DosName  dc.b  'dos.library',0
NewLine  dc.b  $a

end
```

The whole thing

Here's the whole program in one piece. Keep in mind that labels (sometimes called symbols) must be hard against the left hand column for most assemblers to treat them properly. Some assemblers also want comment lines to start at the left margin.

```
; Reverse ECHO program
; Startup sequence, CLI only
; Definitions for Exec Library
   XREF _LVOOpenLibrary
   XREF _LVOCloseLibrary
; LVO means Library Vector Offset
   MOVE.L A0,A5
   MOVE.L D0,D5
   MOVE.L $4,A6              ; Exec base
   LEA    DosName(PC),A1     ; Dos name string pointer
   MOVEQ  #0,D0              ; any version
   JSR    _LVOOpenLibrary(A6)
   MOVE.L D0,A6              ; Dos base pointer
   BEQ.S  Exit               ; Zero, quit
   BSR.S  Main               ; Do the job
   MOVE.L A6,A1              ; Dos base pointer
   MOVE.L $4,A6              ; Exec base pointer
   JSR    _LVOCloseLibrary(A6)
Exit:
   RTS

Main:
; Main Job
; Definitions for Dos library
   XREF _LVOOutput
   XREF _LVOWrite
;
   JSR    _LVOOutput(A6)     ; Get output handle
   MOVE.L D0,D4              ; Handle safe in D4

Loop:
   SUBQ.W #1,D5
   BMI.S  Finish
   LEA    0(A5,D5.W),A0      ; Address of Character
   MOVE.L A0,D2              ; ... to D2
   MOVEQ  #1,D3              ; Length of Character
   MOVE.L D4,D1              ; Handle
   JSR    _LVOWrite(A6)      ; Send character
   BRA.S  Loop
```

```
Finish:
   LEA     NewLine,A0          ; Address of NewLine char
   MOVE.L  A0,D2               ; ... to D2
   MOVE.L  D4,D1               ; Handle
   JSR     _LVOWrite(A6)
   RTS

DosName  dc.b  'dos.library',0
NewLine  dc.b  $a

   end
```

Assembling

Type it all in and save it, using the name OHCE.ASM. Again, be careful about upper and lower case names. No need for a printout yet... the assembler will usually do that for you.

Keep in mind that if you're using AssemPro, you must change the XREF lines in favor of INCLUDE statements, and you will skip the BLINK step described below.

Now start the assembly. On some packages, that means selecting Assemble from a screen menu. Other assemblers will call for you to type a CLI command. You might type:

```
(Metacomco):       ASSEM OHCE.ASM -o OHCE.OBJ
(Charlie Gibbs):   A68K  OHCE.ASM    OHCE.O
```

Note that we specify the source (.ASM) and object (.O or .OBJ) file names; I have not supplied a full path name, since you might choose to put these files in RAM, DF0 or DF1, or even hard disk. To get a listing file, you would expand the commands slightly:

```
(Metacomco): ASSEM OHCE.ASM -o OHCE.OBJ -l OHCE.LST
(Gibbs):     A68K  OHCE.ASM    OHCE.O      OHCE.LST
```

The file names may be anything you wish, but suffixes like .ASM, .OBJ, and .LST help you keep track of which file is which. C.A.P.E. 68K has an *Options* menu that allows you to select names for object and listing files; you may also call the assembler from CLI if you wish. For example:

```
(CAPE 68K):  CAPE -a OHCE.ASM -o OHCE.OBJ -l OHCE.LST
```

The assembler didn't find the values for the _LVO symbols. But that's all right: it will leave a note for the linker.

Linking

We still have only one program 'chunk'; in one sense, our program doesn't need to be linked together with anything else. We do need to fill in the missing _LVO values, however, and we must use the linker in order to get an *executable* file, that is, a program that will run.

A commercial assembler usually has a complete set of 'include' files; these must be licensed from Commodore, and would not be found in freeware. The _LVO addresses you want will be found in a file usually called Amiga.lib.

If you don't have this, there's a freely distributable file called small.lib, produced by Bryce Nesbitt and containing the same information. It may be obtained from club libraries or bulletin boards. You will need to get the _LVO information from one or the other file.

BLink (a successor to the original ALink), by John Toebes and other members of The Software Distillery, is freely distributable and almost universally used for both assembly language and C programming.

The link command usually takes one of the following forms. Again, the full library path is not given here; adjust it to suit your system.

```
(Metacomco):  Blink OHCE.OBJ lib lib/Amiga.lib to OHCE
(Gibbs):      Blink OHCE.O SmallLib/small.lib to OHCE
(CAPE 68K):   Blink67 OHCE.OBJ lib LinkFiles/Amiga.lib
                 to OHCE
```

The file names above are arbitrary, but typical. Keep in mind that C.A.P.E. will produce an object file named OHCE.OBJ only if you ask for it; otherwise you'll get AsmObjTemp. Small.lib works better (with most versions of BLink) if you leave out the LIB (library) keyword.

Running the result

A command in CLI such as OHCE DING DONG will produce the satisfactory response of GNOD GNID. We've 'twisted the tail'.

Here are some things for you to try, or to think over. If we redirect the output, as in OHCE >RAM:XX DING DONG, will the redirection characters be part of the command tail? If so, you'll see them in the output file. Will extra spaces at the beginning or end of the command and its tail be preserved or thrown away?

You might like to verify that RUN OHCE DING DONG works as expected. You could even try executing a script file containing a command OHCE to ensure that everything works that way.

Lastly: if we type OHCE DING DONG, how many characters long will the command tail be? Will the NewLine character at the end of the line be included in the count (in register D0), or not?

Variations and documents

You may have an assembler other than the ones named above, but you'll find that the principles are the same. In case you need to define your own _LVO values, diagrams 1 and 2 give

you the entry point offsets in decimal, hexadecimal, and two's complement hex. Your debugger or disassembler might use any one of these numbering systems.

Don't get swept away buying every Amiga reference text you can find... yet. It would cost a lot of money, and take up a lot of shelf space. Wait until you decide upon the areas you will need to investigate in detail. In the meantime, there's a handy form of quick library documentation that's free to all Amiga owners: the FD files.

The FD files are on your *Extras* disk. They are in text form; you can list them with a CLI command such as TYPE. They will give you a hint as to the fields you will need to set up, and the registers involved.

FDD8	-228	-552	OpenLibrary
FE62	-19E	-414	CloseLibrary
FE80	-180	-384	WaitPort
FE86	-17A	-378	ReplyMsg
FE8C	-174	-372	GetMsg
FEDA	-126	-294	FindTask
FF2E	-D2	-210	FreeMem
FF3A	-C6	-198	AllocMem
FF76	-8A	-138	Permit
FF7C	-84	-132	Forbid

Figure 1: Some entry points in exec.library

FF7C	-84	-132	Forbid
FF22	-DE	-222	Execute
FF7C	-84	-132	IoErr
FF82	-7E	-126	CurrentDir
FFA6	-5A	- 90	UnLock
FFAC	-54	- 84	Lock
FFC4	-3C	- 60	Output
FFCA	-36	- 54	Input
FFD0	-30	- 48	Write
FFD6	-2A	- 42	Read
FFDC	-24	- 36	Close
FFE2	-1E	- 30	Open

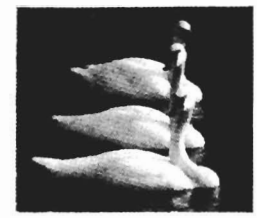
Figure 2: some entry points in dos.library

CygnusEd Professional

by CygnusSoft Software. Published by ASDG Inc.

"The New King of Editors,"
68000er, February 1988

"This editor is definitely one of the best I've used on any machine, and I've used many."
The Amiga Sentry, Sept. 1988



CygnusEd Professional brings unparalleled levels of flexibility and speed to the **Amiga.** features include:

Crash Recovery. Variable screen resolution up to 1000 by 800. Full Macro capability. Over 200 functions and variables available through AREXX. Hot Key operation. Screen refresh at over 30,000 cps. Search and replace at over 100,000 cps. Blitter based horizontal and vertical scrolling. Vertical and Horizontal block cut and paste. Many Word Processing features. Many Programmer features. Edit multiple files on screen at same time.

Asynchronous Printer support. Super interprocess communications facilities. DOS and AREXX commands can be launched from inside the editor. User definable "bookmarks". International keyboard and video support. Mouse or keyboard based Turbo Scrolling. Keyboard and mouse can be interchanged for everything except entering text. Will create icons for WorkBench users. Edits binary files. And **MUCH more!**

ASDG INCORPORATED
925 Stewart Street
Madison, WI 53713
(608) 273-6585

Available At Your Local Dealer **$99.95** U.S. Dollars

CygnusEd Professional Copyright 1988 By CygnusSoft Software. Amiga and Workbench are trademarks of Commodore-Amiga Inc
Developers are encouraged to contact ASDG for possible publication of your products.

SAFE HARBOR SOFTWARE & PERIPHERALS

Great Prices! Shipping based on weight and zone
414 544 2066
414 544-6567

Software

AmigaDOS 1.3	$ 23.25
ATalk III	$ 64.69
ARexx	$ 31.10
Audio Master II	$ 59.10
Aztec C Developer	$195.00
Aztec C Professional	$130.00
B.A.D	$ 33.00
Carrier Command	$ 30.81
CLIMate	$ 27.73
Comic Setter	$ 60.00
CygnusEd	$ 64.69
DeskTop Budget	$ 46.50
Devpac Amiga	$ 64.69
Digi View Gold	$143.10
Disk Master	$ 36.00
DOS 2 DOS	$ 36.39
Dragon's Lair	$ 36.69
Express Paint 3.0	$ 81.50
FA/18 Interceptor	$ 36.69
FACC II	$ 22.00
Falcon	$ 31.00
GOMF 3.0	$ 27.28
Inovatools	$ 52.37
Interfont	$ 76.00
Lattice C 5.0	$190.00
Lights! Camera! Action!	$ 47.90
Magellan	$125.00
Manx Dev/Debugger/Lib	$305.00
Manx Souce Debugger	$ 50.00
Microfiche Filer +	$111.00
Movie Setter	$ 60.00
PageStream	$131.00
Power Windows 2.5	$ 59.09
Professional Draw	$130.00
Project D	$ 32.00
Pro Sound Designer	$111.00
Quarterback	$ 46.21
Superback	$ 47.90
SuperBase Personal 2	$ 90.00
SuperBase Prof. 3.0	$200.00
SuperPlan	$ 99.97
Turbo Silver	$114.00
WordPerfect Library	$ 94.00
WShell	$ 31.10

Abacus Books

Amiga C for Beginners	$ 15.00
Amiga DOS Inside/Out	$ 16.30
Amiga Machine Lang.	$ 16.30
Amiga Systems Progmr.	$ 26.25

Hardware

Amigen	$145.00
CA-880 3.5" Floppy Drv.	$155.00
Electronic Color Splitter	$ 82.00
Floppy Internal 2000	$120.00
Future Sound	$135.00
Live! 2000	$315.00
Midi Gold	$ 67.00
Modem, Supra 2400	$140.00
Mouse Master	$ 31.00
My-T-Mouse	$ 46.77
Processor Accelerator	$145.00
GVP HCard 40Q 2000	$798.00
Quantum 40Q SCSI HD	$580.00
Spirit Brd 500/1000 0K	$212.56
Spirit Brd 500/1000 512K	$425.00
Supra SCSI Cntrlr 2000	$205.00

Orders Only Please:
Outside WI: **800-544-6599**
Inside WI: **414-544-6599**
Visa/MC/CODs

2414 Pendleton Place ■ Waukesha, WI 53188 ■ 9 AM to 5 PM M-F

BreakPoint Part Five

And finally, a look at DB and MetaScope

by Victor A. Wagner
Copyright © 1989 Victor A. Wagner, Jr.

Vic Wagner started his association with computers in 1965 while in the US Air Force working in a group studying digital flight simulation. Since becoming a civilian in '66, he has been working mostly with minicomputer manufacturers in the real-time systems software area. Weekdays from 8 to 5 find Vic answering the technical support line for Computer Automation Inc., and maintaining real-time software systems. Evenings and weekends he spends most of his waking hours talking to Taarna (his Amiga 1000), writing programs, and spending time on CompuServe's Amiga forums (76046,3004) where he is currently teaching a C class.

In this last instalment we're going to take a look at the remaining two readily available debuggers in the Amiga marketplace. These two are the old-timers in the debugging arena for the Amiga. They are also about as different in user interface as two programs could possibly be. DB, from Manx Software Systems, uses primarily a text interface. MetaScope, from Metadigm, is completely 'Intuitionized' (menus, gadgets, point and click). The two programs are broadly equivalent in abilities.

Before we get into the specifics and the differences, I should point out that I have been using MetaScope since before it was publicly released, and that I have some financial interest in its success as well as a fair amount of emotional interest in it. I've known the author for over ten years; throughout MetaScope's design and implementation, we spent many a lunch hour chowing down pizza (the only *true* food of programmers) and discussing features and options. I have done extensive 'alpha' and 'beta' testing of MetaScope and am the author of the demonstration text on the current release. That MetaScope follows (to some extent) my prejudices in debugging techniques is therefore not surprising, although some of its 'features' drive me to distraction. I will endeavour to give as unbiased a view as possible.

Okay, let's look at some of the 'physical' aspects of the debuggers before we enmesh ourselves in the features of the programs themselves.

```
MetaScope 81336 ----rwed 17-Jan-89 11:32:06
db        60360 ----rwed 18-Jun-87 17:30:48
```

The versions on my system are:

Aztec C Debugger - Version 3.4b - 6-16-87
MetaScope: The Debugger (Version 1.23)

This version of MetaScope is a 'beta' version that has source capability with the Lattice compiler; by the time you read this, it 'should' be working with the Manx compiler as well.

As you can see, MetaScope is about 35 per cent larger on the disk. In memory they look like this:

MetaScope:	size		DB:	size	
address	Hex	Dec	address	Hex	Dec
0267D8	D4	212	86EBF8	4	4
87A5D0	3398	13208	87A5D0	2D4C	11596
8CDAC8	103C0	66496	87D328	31B4	12724
			8921B0	32D0	13008
	1382C	79916	895B70	B14	2836
			89E8D0	22C8	8904
			8B80C0	24CC	9420
			8BA598	2028	8232
				104A4	66724

MetaScope is about 20 per cent larger when in memory. Remember that the Amiga does scatter-loading, so the addresses have little meaning. The other physical differences are in the manuals. DB's manual is 42 pages 3-hole punched, and included in the binder with the Manx compiler. MetaScope's manual is a 35-page centre-stapled booklet 5-1/2" by 8-1/2". Neither manual has an index, but both have tables of contents. Both manuals are 'written in the style' of computer program manuals. This is to say that each is more like a dictionary than a novel. Considering the difficulty I am having writing about debugging in general, I have a great deal of empathy with the authors of these manuals. There is a lot of information to impart to the reader, and most of the information is rather diverse and disconnected. All debuggers seem to be collections of things that programmers have found useful over the years. This should not be taken as negative criticism of the manuals, but rather as an explanation for their somewhat disjointed text.

Even casual perusal of the manuals will point out the vast difference in approach to operator interface. DB is strictly textual, like most of the debuggers that preceded it. It opens with a single window containing its log-on message and a prompt waiting for a command. The usual way to gain control of a task in DB is by using the 'al' or 'aL' command. This puts some hooks into AmigaDOS and grabs onto the next program to be executed, either from the CLI or the Workbench. After DB gains control, it prints the registers and awaits further commands. MetaScope, on the other hand, opens up what it calls a status window and then just sits there. There is not even any place to type. MetaScope, however, is a very Amigatized program; in its case you load a program by making a menu selection. MetaScope also allows you to put the 'command line' for the program you're debugging on the MetaScope invocation line as in:

```
1>MetaScope df0:projs/aargh -a -b -c
```

The above example was taken directly from the MS (as I'm going to call MetaScope henceforward) manual. DB, on the other hand, has a 'default file' that it will read upon invocation... if it contains the single command 'al' (Amiga, load with symbols) then the following:

```
1>db
1>df0:projs/aargh -a -b -c
```

will do the same thing as the MS invocation.

Rather than give you a blow by blow of what each debugger can do, I'm going to tell you the things each can do that the other cannot. Otherwise we're going to end up with a very long article that will only serve to confuse the reader (as well as the writer). By now, you've seen or played with at least one debugger, so I'm sure you're relieved that I'm not going to repeat the above few paragraphs for each and every 'command'.

Both debuggers have rather complete expression evaluators. The MS manual takes 2-1/2 pages to detail its expression capabilities (which resembles those of the C language). DB takes about 3 pages. I wouldn't bet a nickel on either of them being more powerful than the other.

Okay, now on to the specifics. I'll go through the DB commands in roughly alphabetical order and comment on those things missing from MS.

Gee, I didn't get very far at all into the DB command list. DB has many Amiga-specific commands (they all start with 'a') for which there are no counterparts in MS. Many (mayhap all) of the exec lists can be displayed: devices, interrupts, libraries, ports, resources and tasks.

One of the 'a' commands is particularly useful, especially if you are debugging multiple tasks. This is the 'an' command. When DB is started it opens one window for dialogue. With this command you open another window and you can now debug two tasks at the same time. Of course, you use the 'al' command to capture the next task to start. Alternatively, you can use 'as' to halt and begin debugging a task that is already running, or even 'ap' to debug, and possibly rescue, a task that is held because of an exception ('software error') such as an address error or a divide by zero.

DB has breakpoints that 'watch' the value of memory locations, and will then stop execution if the memory either becomes equal to (or unequal to) a specified value. The length of time it takes DB to recognize that the memory location has changed depends on the mode DB is in at the time. If you are single stepping, the detection is immediate; if not, these memory change breakpoints will be examined at each function entry and exit. DB also has the ability to 'hash' or checksum a region of memory and to check this periodically as above. Explicit warning against doing this to the first 256 bytes of memory is given since this would slow down the ability to single step. Note that there is a special command to watch the first 256 bytes of memory.

DB has one breakpoint feature I'm sure every programmer has wanted at some point. You can set the address of a function that DB will call *every* time it regains control. If the function returns zero, then it is as if nothing had happened. If, however, the function returns a non-zero value, the program is stopped, and a message is displayed showing the value returned.

DB allows two temporary breakpoints to be set, and allows temporary breakpoints either with or without the breakpoints in the breaklist being activated.

It also allows a 'command list' to be executed upon reaching an execution breakpoint. The most common use of this feature is to display certain memory locations when the progam reaches the breakpoint.

There are also commands for allocating (for patches), comparing, filling, and moving memory.

DB devotes almost six pages to the 'print' command. This command allows you to format data in almost any conceivable fashion. The allowable formats seem to be a superset of printf(), and at least as cryptic.

You can define up to 26 macros to be saved (one per letter of the alphabet). This should save a fair amount of time, as reading keys from memory is much faster than typing.

DB can 'redirect' its I/O. Commands can be input from a file, and the output that goes to the window can be directed to a file also. Especially useful is the ability to log *commands only* to a file, so that you can read them in again later, and retrace the steps being done now.

The first thing you notice when you start playing with MetaScope is that there appears to be no limit to the number of output windows that may be opened. This is broadly true (though

since each window takes *some* memory, there is a limit based on available memory); however, the usefulness of having multiple 'break', 'hunk', 'status', or 'symbol' windows is limited.

Having multiple 'memory' windows, on the other hand, alters one's method of debugging. You may be used to supplying commands to display some memory locations each time a breakpoint is reached. That is uneccessary in MS (it's impossible as well). You simply open a window on the requisite memory, and MS will display it.

Before you say, "Wait a minute, I use my commands to display things based on register (or variable) contents", I should point out that MS allows its full expression capability to determine the start address of a window. This expression can also be either 'static' (determined at the time it is entered) or 'dynamic' (determined each time the window is to be displayed). Particularly useful expressions are:

pc (the current program counter... useful for seeing where the program currently is)

sp (the current stack pointer... for looking at all those arguments to functions, and whatnot)

Of course, you'd want these expressions to be 'dynamic' so that you can see the memory as the registers change. (The use of the word 'dynamic' might be slightly misleading since the expressions that contain registers are evaluated only when MS has control of the program. While this allows you to look at memory referenced by registers, it doesn't waste a lot of time redisplaying different regions of memory when your program is actually running.)

"But certainly, MS doesn't display changing memory while my program is running, and MS isn't in control", I hear you ask. Well, that may be the biggest difference between MS and the other debuggers we've looked at.

MS updates its memory displays around ten times a second independent of whether the program is running or not. This means that you can look at such things as the mouse x and y variables in a Window structure and watch them change as you move the mouse around.

MS has scroll bars and sizing gadgets on all of its windows so that you can change the amount displayed as well as scroll around. You can select addresses for breaking (or just examining) by pointing and clicking on them in the windows.

'Conditions' can be placed on permanent breakpoints. These are expressions that are evaluated whenever the associated breakpoint is reached. If the expression is 'true' (non-zero), the counter associated with the breakpoint is incremented and the break actually occurs if the counter has reached the desired number. This means that MS can produce breakpoints like: "break the third time you get here and the D0 register is equal to the contents of the beginning of the input buffer".

A 'freeze' feature is provided for both status windows and memory windows. The freeze function keeps the contents of the window constant for later comparison with the current state. MS can also restore the machine to the state contained in a 'frozen' window. This means that you can 'save the state' of the machine (by freezing the appropriate windows) then restore it later if you want to 'go back and try it again'.

MS has a log capability. You can have it automatically record all of its actions in a file (or to the printer). You can also 'dump' a window to the log.

You can change the title on any window. When you have ten windows out there pointing at memory, sometimes it's tricky to keep track of which is which. Titling them with such imaginative things as "Where we are now" (for a window whose dynamic base address is 'pc'), or "Stack" (for a dynamic base of 'sp'), can be a lifesaver. And when MS writes such a window to the log, the title is included.

You may change a value in memory (or a register) just by double-clicking on it; this invokes a requester where you type in a new value.

You may also change an instruction in memory - by typing in assembly format. If you have a memory window open that is displaying in code format rather than data format, instructions may be 'double clicked' and altered as described above. This means that you don't have to carry a complete list of 68000 opcodes around your desk when you go to patch your program.

A couple of features are available only in the newest (and as yet unreleased) update of MetaScope. The author assures me that this update will be available in March of 1989.

MS allows you to save a configuration of: permanent breakpoints; window locations, sizes, base address expressions and status (frozen or not). This allows you to set up your debugging session as you like, then save it so you don't have to retype it all in each time you start a new session.

You can (again, in the upcoming version) intermix source display with the disassembly in memory windows. So now you can 'see' what your compiler is doing. Clicking on a source line is equivalent to clicking on the first address of generated code for that line.

Well, that about wraps up the differences between the debuggers. Whether you like windows and point-and-click over a mostly text environment is personal preference. I think having memory continuously displayed is an extremely handy feature, but I'm sure there are others who would be more attracted by DB's capability of executing 'command lists' at each breakpoint.

I hope you've enjoyed reading this series of articles. I've had fun writing them. Perhaps after MetaScope is released with

full source capability, and Avant-Garde gets its M2 debugger ready, we can again take a look at the latest features (I presume that Lattice and Manx will have updated CPR and SDB by then) and ideas in debugging.

I've tried to be not too dogmatic in this series. My experience with many systems, debuggers, and programmers is that each person has to pick a style with which he or she is comfortable. Debugging is still an 'art' in many senses of the word, and if *you're* comfortable with the technique, then it will work for you. If you're *not* comfortable with the technique, then it will just get in the way of you getting some work done.

I would like to leave you with one maxim that has helped me in my debugging more than any other. It is:

Have *some* idea of what you expect *before* you run the test.

He sprang to his keyboard
One last break to put in
But he started to vanish
He was looking quite thin.

And I heard him exclaim
'ere he faded from sight
"Happy debugging to all...
Give that GURU a fight!"

The Disk Mechanic

Give your disks a Tune-up

Only $89.95

The Disk Mechanic is a comprehensive collection of utilities for Amiga DOS, including TuneUp, DoubleBack, DiskRepair, and WorkShop.

TuneUp: With TuneUp you can improve hard disk or floppy disk performance by up to 400%. Not a caching program, TuneUp is a sophisticated disk optimizer that actually re-organizes the information on your disk to improve system efficiency. TuneUp is fully compatible with Amiga DOS.

DoubleBack: DoubleBack is a high speed archival hard disk back-up program. DoubleBack supports multiple source and destination drives, file selection by name, date, protection, directory, and file note. Back up a full 20 megabytes in under 40 minutes. Selectively copy files between hard disks and partitions. Protect your data!

Sample TuneUp benchmark of some standard 3.5" disks:

Volume Name	TuneUp	BEFORE	AFTER	IMPROVED
Workbench 1.3	3:30	4.86	2.70	79%
Digi!	3:45	9.44	5.22	80%
WordPerfect	4:42	12.36	5.37	130%
SuperBase	3:37	19.39	9.66	100%
data	1:43	14.30	4.68	205%
archives	3:40	28.43	9.52	299%

* Times given are the average of three runs on a standard A2000 internal floppy drive. The TuneUp time is the time required to process the disk (in minutes and seconds). The BEFORE and AFTER times are the time (in seconds) required to get a directory of the disk. The more use a disk has had, the better the improvement!

Demo disk Available; call for details.

DiskRepair: With DiskRepair it is possible to recover accidently deleted or discarded files, salvage files from damaged disks, and reconstruct a corrupted hard disk. DiskRepair works with hard and floppy disks, and is compatible with the Fast Filing System.

WorkShop: For the expert user, the disk WorkShop is a full featured disk file and block editor. Examine and modify disk files, search for viruses, and change menus or requesters in popular programs.

The Disk Mechanic also includes a dozen small CLI based utilities to make managing your disks easier. We are now shipping version 2.5 with full support for Amiga DOS 1.3 and the Fast Filing System. The Disk Mechanic is not copy protected.

Dealer inquiries welcome.
Amiga is a trademark of Commodore Amiga Inc.
WordPerfect is a trademark of WordPerfect Corp.
Digi! is a trademark of Aegis Development
SuperBase is a trademark of Precision Software, Ltd.

Lake Forest Logic Inc.
28101 E Ballard Road
Lake Forest, IL 60045
(312) 816-6666

MasterCard VISA

World Class AUI!!!

Now in North America!

AUI - Amiga User International Magazine - is now available in the U.S.A. and Canada. Amiga User International, established for over two and a half years, was the first Amiga magazine in Europe, becoming the Number One publication on the International Amiga scene.

Published by Croftward, in London, England, also the publishers of Transactor for the Amiga, it brilliantly covers the whole range of Amiga Users' interests from programming to games. Its articles and series are written by some of the most expert global commentators on the Amiga and the fast turnaround it achieves means the latest products are knowledgeably reviewed and the hottest news appears on its pages. Amiga User International Magazine is now available in North America, either on the newsstand or by subscription.

Look out for AUI at your local book, magazine or computer store. If you would like to subscribe for the exciting, informative world-class Amiga magazine, then fill in the coupon below for a special North American introductory subscription offer.

If you do not wish to cut up your magazine, a photocopy will be welcome.

To: **Subscriptions Manager, A.U.I.**
85 West Wilmot St., Unit 10
Richmond Hill, Ont. L4B 1K7

Please send me 12 issues of Amiga User International at the special subscription rate of

☐ U.S. $30.00 ☐ Cheque Enclosed
☐ Canada $37.00 ☐ Visa/Mastercard

Credit Card Acct #: Exp. Date:

Name _____

Address _____

Tel. No. _____

BRiWALL

Our Fourth Year
In Business!
We Know How
To Service
Our Customers!

SOLID PRODUCTS & SOLID SUPPORT

P.O. Box 129 / 58 Noble Street
Kutztown, PA 19530

24 HOURS — TOLL FREE
1-800-638-5757

OUTSIDE USA CALL 1-215-683-5433

SATISFACTION
GUARANTEED

No Surcharge On
Charge Orders

Friendly Service

Customer Service
Business Hours
M-F, 9-5 EST

BOOKS

1001 Things to do with Amiga	$ 11
AMIGA C for Beginners +	15
Advanced Amiga BASIC Book	15
Machine Lang Prog Gde/Compute	17
Amiga BASIC Inside&Out Book +	20
Amiga DOS Inside&Out Book +	17
Amiga DOS Manual	20
Amiga DOS QUICK Ref Manual	12
Amiga DOS Reference Manual	19
Amiga DiskDrives Inside&Out +	24
Amiga Machine Language Book	17
Amiga Programmer's Guide	15
Amiga Progrmr's Handbk Vol1	20
Amiga Progrmr's Handbk Vol2	20
Amiga System Progrmr's Guide	27
Amiga Tips & Tricks Book	17
Amiga for Beginners	15
ROM Kern Ref Incids & Auto*	27
ROM Kern Exec & Lib & Devices*	28
Hardware Manual*	18
Elementary Amiga BASIC Book	13
Inside Amiga Graphics	15
Inside the Amiga With C	21
Kickstart Guide to Amiga	19
+ Disks for Books	12

GENERAL BUSINESS

Acquisition v1.3	$189
Analyze 2.0	69
Becker Text	99
Citydesk v 2.0	129
Critic's Choice	169
Data Retrieve	52
Data Retrieve Professional	199
Easy Ledger*	199
Excellence	169
Family Tree (Your)	35
Flow	65
Haicalc	39
KFS Accountant-Amiga	195
Keepin Track	45
Kindwords	69
Logistix	69
Magellian	139
MaxiPlan Plus	125
MaxiPlan 500	95
Microfiche Filer	69
Microfiche Filer Plus	125
Microlawyer	39
Money Mentor 2.0	69
Nimbus Accounting	89
Organize	49
Pen Pal*	99
P.H.A.S.A.R.	69
Page Setter	90
Page Stream	135
Prowrite v 2.0	75
Postscript for Prowrite	32
Professional Pages 1.1	249
Publisher Plus	64
Reason	249
Scribble	55
Security Analyst	65
Shakespeare	149
Softwood File II SG	69
Softwood Write & File	69
Superbase Personal 2	105
Superbase Professional	235
Text Ed Plus	59
Tax Plan 1988/1989	28
Tax Break	59
Textpro	59
Word Perfect	189
Word Perfect Library	99
Works (The)	125
Works - Platinum Edition*	175

Products marked with * were not released at time ad was prepared. Please call for firm price and availability.

EDUCATIONAL

Adventures of Sinbad	$ 32
Aesop's Fables	32
All About America	39
Animal Kingdom (Wonders of)	32
Decimal Dungeon	32
Dinosaur Discovery Kit*	29
Discovery Math	27
Discovery Spell	27
First Letters/Words	30
First Shapes	30
Fraction Action	32
Ghostly Grammer	32
Kid Talk	30
Kinderama	32
Linkword: French	24
Linkword: German	24
Linkword: Italian	24
Linkword: Spanish	24
Mad Libs	17
Math Talk	30
Math Talk Fractions	30
Math Wizard	32
Math-amation (College)	55
Mavis Beacon Typing	36
Perfect Score SAT	55
Planetarium*	49
Puzzle Storybook (The)*	29
Read & Rhyme	32
Read A Rama	32
Smooth Talker	30
Speller Bee	29
Surgeon	34
Word Master	32

ENTERTAINMENT

4x4 Off Road Racing	$29
AAArgh	25
AD&D Heroes of the Lance	29
Annals of Rome	27
Arkanoid	20
Artic Fox	17
Awesome Arcade Pak	36
Balance of Power	36
Barbarian	28
Bard's II: Destiny Knight	39
Bard's Tale I	33
Bard's II: Clue Book	12
Battle Chess	39
Blackjack Academy	27
Breach	27
Bridge 5.0	25
California Games	29
Capone	27
Captain Blood	33
Carrier Command	32
Centerfold Squares	24
Chessmaster 2000	36
Chessmaster 2100*	36
Chrono Quest	35
Clever & Smart	22
Clue: Master Detective*	29
Craps Academy	32
Defender of the Crown	33
Double Dragon*	29
Dragon's Lair	35
Dungeon Master	28
Earl Weaver Baseball	35
Earl Weaver's Commisioner's Disk	$ 17
Empire	36
F18 Interceptor	36
Faery Tales	32
Faery Tales Clue Book	9
Falcon	35
Ferrari Formula One	36
Fire & Forget	29
Fire Power	19
Flight Simulator 2	35
Fourth & Inches	29
GB Air Rally	25
Galactic Invasion	19
Galileo 2.0	49
Go	28
Gone Fish'n (Reel'n Fish'n)	35
Guild of Thieves	29
Gunshoot	22
Hardball Baseball	29
Hole In One Miniature Golf	26
Hunt for Red October	36
Hybris	27
Ice Hockey-Super Star	35
Ice Hockey: Wayne Gretsky	35
Jet	35
King of Chicago	33
Kings Quest III	36
Knight Orc	29
Lance-A-Lot	29
Land of Legend	32
Leaderboard Dual Pak	26
Leaderboard World Class	29
Life & Death (surgical game)*	35
Lord of the Rising Sun	33
Marble Madness	3f
Master Ninja	29
Mean 18 Golf	27
Mobieus	39
Obliterator	28
Out Run	37
POW	29
Phasar Gun	42
Pioneer Plague	29
Platoon	33
Port of Calls	32
President's Missing (The)	32
Questron 2	36
Reach for the Stars	29
Return to Atlantis	36
Rocket Ranger	33
Roger Rabbit	33
Romantic Encounters - Dome	27
Rubicon Alliance*	24
SDI	33
Sargon 3	29
Scrabble	29
Sex Vixens from Space	29
Sex Vixens Hint Sheets	5
Shadow Gate	33
Shangai	27
Sky Fox 2	29
Space Quest 2	36
Space Station Oblivion	36
Spacecutter	25
Star Fleet 1	39
Star Glider 2	32
Stellar Conflict	27
Strip Poker 2	29
Sword of Sodan	34
TV Sports Football	33
Terror Pods	28
Test Drive	27
Thexder	$ 25
Three Stooges	33
Time & Magic	29
Tracers	27
Ultima IV	39
Universal Military Simulator	35
Vampire's Empire	29
Video Vegas	25
Virus	25
Wizard Wars*	29
World Tour Golf	29
Zach Macrakin	32
Zany Golf*	33
Zoom	20

CREATIVITY/GRAPHICS

Analytic Art	40
Animate 3D	99
Animation Effects	39
Animation Flipper	45
Animation Stand	39
Animator	69
Award Maker Plus	35
Business Card Maker	37
Calligrapher v 1.05	79
Celebrity Cookbook	27
Comic Setter	69
Comic Setter Clip Art each	25
Deluxe Paint v3.0	109
Deluxe Photo Lab	99
Deluxe Print 2	55
Deluxe Productions	139
Deluxe Video 1.2	86
Design Disks	each 25
Digi-paint	40
Digiview 3.0	129
Digiview Gold for A500/A1000	129
Director	49
Draw 2000 (formerly Draw Plus)	159
Express Paint V3	89
Fantavision	49
Flipside	25
Graphics Studio	39
Home Builder's CAD	135
Icon Paint	29
Images	29
Intro CAD	52
Invision by Elan Design	99
Lights, Camera, Action	59
Masterpiece Amiga-Fonts/Clip Art	165
Media Line Font Disk 1	24
Modeler 3-D	69
Movie Setter	69
Page Flipper	39
Page Flipper Plus FX	99
Photon Paint	65
Photon Paint Surface Disk	25
Photon Video Cel Animator	99
Photon Video Transport Controller	195
Pixmate	45
Printmaster Art Galleries each	22
Printmaster Plus	33
Pro Video Plus	199
Sculpt 3D	69
TV Show	65
TV Text	65
Turbo Silver Plus	139
Ultra CAD*	169
Video Scape 3D 2.0	125
Video Titler 1.1	99
X-Specs	$ 92
Zoetrope	89
Zuma Fonts	each 25

SOUND/MUSIC

Deluxe Music Construction	$ 65
Dr. T Song Book	20
Dr. T Copyist II	219
Dr. T Drums	20
Dr. T KCS v16a	165
Dr. T Keys	20
Dr. T MT32 Voices	39
Dr. T Midi Recording Studio	49
Dynamic Drums	52
Dynamic Studio v2.4	135
Instant Music	36
Midi Magic	99
Model A Midi Interface	49
Music-x*	195
Perfect Sound A1000	65
Perfect Sound A500	65
Pro Sound Designer	115
Sonix	52
Sound Oasis	65
Sound Sampler	85
Soundscape Pro Midi Studio	129
Studio Magic	65
Synthia	65
Texture	99

UTILITIES/LANGUAGES

64 Emulator 2/Amiga 500	$ 52
64 Emulator for Amiga 1000	52
A Talk Plus	52
AC/BASIC v 1.3	135
AC FORTRAN	199
Amiga Workbench 1.3	24
Amikit	34
Assempro	64
Aztec C Developer Compiler	195
Aztec C Professional Compiler	129
Aztec C Source Level Debugger	49
BBS PC	99
Benchmark Modula 2	129
Benchmark Modula 2-Simp A Lang	69
Benchmark Modula 2-IFF Library	69
Butcher 2.0	25
C.A.P.E. 68K v2.0	59
CLI Mate	27
Cygnus Ed	69
DOS-2-DOS	39
DSM	49
Developer's Pak 2	69
Diga v1.0	52
Disk-2-Disk	35
Diskmaster	39
FACC II	25
GOMF V3 with Hardware Switch	49
Gomf v3.0	27
Grabbit	20
Inova Tools	52
Lattice C + +	339
Lattc C MS-DOS/AmigaDOS Complr	359
Lattice C V 5.0 Developer Pack	205
Mac Emulator*	call
On Line V2	42
Power Windows 2.5	59
Project D v1.1	35
Quarterback	45
True BASIC Precalculus Library	35
True BASIC 3D Graphics Library	35
True BASIC Advanced String Library	35
True BASIC Algebra 1 Library	23
True BASIC Algebra 2 Library	35
True BASIC Calculus Library	35
True BASIC Discrete Math Library	35
True Basic 2.0	69
UltraDOS	39
Virus Infection Protection	33

AMIGA

IT IS OUR POLICY to stock what we advertise and carry the best products available at the best prices possible. OVER 700 PRODUCTS in stock now and more arriving daily!! IF YOU DON'T SEE IT LISTED, PLEASE GIVE US A CALL.

HARDWARE

Let us put a custom package deal together for you!

Please Call M-F, 9-4 EST

GENERAL

AmiGen Genlock	$149
CMI VI-500 Video Adapter for A500	68
CMI VI-500RF Video Adapter for A500	88
Color Splitter	75
DRAM 256Kbx4-120ns	41
DRAM 1Mbx1-100ns	35
DRAM 256Kbx1-150ns	11
32MB 3.5" Seagate ST-138N-0	439
48MB 3.5" Seagate ST-157-N-0	565
64MB 5.25" Seagate ST-277N-0	525
40MB 3.5" Quantum ProDrive	589
80MB 3.5" Quantum ProDrive	990
Easyl 1000 Draw Tablet	369
Easyl 2000 Draw Tablet	369
Easyl 500 Draw Tablet	329
Flicker Fixer A2000	495
Gen/One Genlock	$ 719
Live! 1000	249
Live! 2000	335
Live! 500	285
Magnavox 8CM-873 Multisync Monitor	579
MIDI-Internal for A2000	75
Mitsubishi AUM1381A Monitor	579
Panasonic 1410 Camera	189
Rainbow 16mm lens w/variable iris	49
Perfect Vision	169
Processor Accelerator by CMI	169
SCSI Module for Starboard II	99
Spirit Inboard for A1000 OK	240
Starboard II OK	269
Subsystem 500	199
SuperGen	689

Actual Shipping Costs on Hardware

AMIGA

1084S Stereo Monitor	call
2058 Memory Board w/2MB (8MB max)	call
2090A Autoboot Controller	call
2300 Genlock for A2000	call
Amiga 68020/68881 w/2MB	call
Amiga 500	call
A501 Memory Expansion for A500	call
Amiga 2000, 2000HD or 2500	call

DIGITRONICS

RC4 Ramcard OK	$ 199
RC4 Ramcard for A500 complete	280

GREAT VALLEY

A2000 SCSI Hrd Dsk Cntlr 2MB/OK	299
A2000 2/0 SCSI Cntr w/80MB Quantum	1249
A2000 30MB SCSI Hard Card	679
A2000 40MB SCSI Hard Card	799
A2000 45MB SCSI Hard Card	789
A2000 80MB Quantum SCSI Hard CArd	1159
A500 30MB Hrd Drive/RAM Sbsystem	759
A500 40MB Qntm Hrd Drv/RAM Sbstm	1020
A500 45MB Hrd Drv/RAM Sbsystem	959
A500 80MB Hrd Drv/RAM Sbsystem	1429

MAST

3.5" Internal Drive for Amiga 2000	129
Tiny Tiger 20MB	579
Tiny Tiger 30MB	629
Tiny Tiger 40MB	799
tiny Tiger 50MB	769
Tiny Tiger 80MB Quantum	1129

PHOENIX

Expansion Chassis A500 or 1000	$ 175
Expansion Chassis w/pwr A500 or 1000	253
Hard Drive 22MB A500 or A1000	579
Hard Drive 48MB A500 or 1000	829

PRINTERS

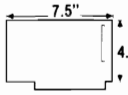
Canon PJ1080A Color Ink Jet	519
Hewlett Packard Desk Jet	779
Hewlett Packard Paint Jet	1150
NEC LC890 Postscript Laser	3499
Okimate 20	155
Star NX-1000R Rainbow Printer	269
Xerox 4020 Color Ink Jet	1139

PROGRESSIVE

ProGen Genlock	379
ProRAM 2000 OK (8MB max)	249
ProRAM 2000 2MB (8MB max)	639
Vault 20MB	569
Vault 30, 43, or 65 MB	call

TRUMPCARD™

THE ACE OF HARD DISK CONTROLLERS



Amiga 2000
SCSI Controller
$159

Amiga 500
SCSI Controller
& Case
$219

TRUMPCARD INCLUDES:

Utilities Program Disk
2 Foot 50 Pin SCSI Connector
Autoboot, Eprom, Complete Manual &
One Year Warranty on Parts & Labor

Trumpcard with 32MB 3½"	$499
Trumpcard with 48MB 3½"	$619
Trumpcard with 62MB 5¼"	$649
Trumpcard 500 with 32MB 3½"	$559
Trumpcard 500 with 48MB 3½"	$679
Trumpcard Hardcard 40MB	$799
Trumpcard Hardcard 100MB	$1159

AMIKIT

Introduction to the Workbench and CLI.

If You:

-are having trouble using the Workbench
-are discouraged because a CLI doesn't obey you
-are running programs only from the Workbench
-are not able to run a program without an ICON
-have not tried multitasking
-have not used ED
-have not modified your startup-sequence
-have not used your RAM disk
-are unfamiliar with the "Know Your Path" rule

Then the AmiKit is for you! It includes an informative 160 page book to help you start using the Workbench, the CLI, and a modem; two disks containing powerful public domain programs such as wordprocessor, a communications program, a spreadsheet, a database, disk utilities; a third disk contains the latest, official version of the Workbench.

OUR PRICE $34.00

The Best Deal in Town! MAST presents the MATCHBOX Collection

Superslimline External Drives

880K External Floppy Disk Drives, just one and two inches high respectively. Compatible with ALL Amiga Computers; Switch deselectable to disable drive(s); No Click; 3 Msec Step Time; Quality Fujitsu Drive; Special Circuitry which draws minimal power from your Amiga permits cool and reliable operation; Pass Thru on the Twindrive; Complete with a Twelve Month Warranty! OUR BEST SELLING EXTERNAL DRIVES!!

UNIDRIVE-$149 TWIN DRIVE-$259

AFFORDABLE, EXTERNAL MEMORY EXPANSION

In a TINY external case, which you simply plug into your A500 or A1000, you get up to 2 megabytes of RAM with full Auto-configure architecture, low-power CMOS design, RAM tachometer, CONFIGURE indicator and more. These things are HOT!

Minimegs (specify A500 or A1000)

512K-$325 1MB-$485 2MB-$805

Coming Soon! MICRO MEGS JUST $165.00

! SAVE $250 off of list !

VIDEO PACKAGE SPECIAL

Panasonic 1410 Camera
16mm Lens with Variable Iris
Copystand with Lights
Digiview Gold

PACKAGE PRICE—$415

(Offer good until 5/5/89)

BRiWALL

SOLID PRODUCTS & SOLID SUPPORT
PO Box 129/58 Noble Street
Kutztown, PA 19530

Toll-Free 24 Hour Order Line
1-800-638-5757

24-Hour Order Line Outside USA
1-215-683-5433

24-Hour FAX Line - 1-215-683-8567

Customer Service - 1-215-683-5433

Technical Support - 1-215-683-5699

PLEASE COME VISIT US!
Monday - Friday 9AM-6PM EST
Saturday 9AM - 12 Noon EST

VISA

Write or Call for our GREAT AMIGA OR C64/C128 CATALOG

ACCESSORIES

WE CARRY A FULL LINE OF

Covers
Printer Ribbons
Printer Ink
Modem Cables
Printer Cables
Video Cables
Joysticks
and more!

CALL FOR PRICES

ECE Midi 1000 or 500/2000	$49
Copy Stand with Lights	59
Flicker Master	14
Inkwell Lightpen	89
MiniModem / A1000	79
MiniModem / A500	79
Phoenix A500 Power Supply	85
Supra 2400 Modem	139

SHIPPING POLICY
All ORDERS received before 3PM EST will normally be shipped same or next business day. Out Of Stock Items will be shipped by the same shipping method as original order, normally within 3 or 4 business days. All UPS shipments are sent SIGNATURE REQUIRED/NO DRIVER RELEASE. ADD the following shipping charges to your TOTAL software order:
UPS: Ground $4.00 (cont USA ONLY); Air/Rush $7.00 (includes Puerto Rico/Hawaii); Overnight $12.00 + $2.50 per item (must be received by 12 NOON)
PRIORITY MAIL: USA $4.00 (includes APO/FPO); Canada/Mexico $6.00; Other Foreign $4.00 Handling and Actual Shipping (minimum $12.00)

ORDERING INSTRUCTIONS
For your convenience, you can place a phone order 24 HOURS/7 DAYS A WEEK on our TOLL-FREE LINE 1-800-638-5757. Outside the USA, please use (215) 683-5433. Price, Availability and Compatability Checks are also welcome on our order line, Monday thru Friday, 9:30AM -4:30PM EST. AFTER HOURS, Orders Only, Please!! When placing an order, please specify your Computer Model, Home & Daytime Phone Number, Shipping Address, Charge Card Number, ExpirationoDate, & Name of Bank. To help us serve you better, please have all your information ready before you call us.

ACCEPTED PAYMENT METHODS: We welcome payment by PREPAID (by Personal Check or Money Order); MASTERCARD; VISA; SCHOOL PURCHASE ORDERS. COD orders are welcome for established BriWall Customers Only ($4.00 additional). All payment must be in USA DOLLARS. There is no Surcharge for Credit Cards and your card is NOT charged until we ship!

RETURN POLICY (for software and accessories ONLY)
We have a liberal return policy to better service your needs. If within 15 days from the time you receive an item, you are not satisfied with it for any reason, you may return it to us for either a refund, exchange or open credit. REFUNDS ARE SUBJECT TO A 10% RESTOCKING FEE PER ITEM RETURNED ($5.00 MINIMUM PER ITEM). EXCHANGES/OPEN CREDITS will gladly be issued for the Full Purchase Price of the item. DEFECTIVE Items, Hardware Items and Special Order Items will be replaced with the same item only.

TECHNICAL SUPPORT
We do our very best to help you with your product selections, before you order and after you receive your product. General questions, price, compatibility with your computer, etc. will be handled by our order staff at the time you place your order. BUT, if you have specific, detailed questions about a product, printer, compatibility questions etc., you will get the most help from our TECHNICAL SUPPORT LINE at 215-683-5699. Call Monday thru Friday, 9:30AM-4:30PM EST, and our trained tech staff will be happy to help you.

Quirks And Quandaries

A collection of bug reports, problems and solutions

by Betty Clay

Betty Clay, a former mathematics teacher, chose writing for her second career. She is the Amiga columnist on STARTEXT (a video service of the Ft. Worth Star-Telegram). She has a regular column with ICPUG, and her work has also appeared in The Amigan. Betty can be reached via CompuServe (76702,337), QuantumLink (bjc), or by mail at 1322 South Oak Street, Arlington, Texas, 76010.

Is there such a thing as a bug-free computer? I doubt it. Can even one piece of software be totally free of them? Not likely. Little faults cause us trouble, and as we find them, one by one they get fixed, or we discover ways to work around them. This article describes some of the strange or troublesome things that have been reported recently. A few of these came from personal experiences, but most have come from others. They were collected from various networks, magazines, newsletters, and conversations. Perhaps they will be of interest, or maybe of help, to you.

SetComment

Matt Dillon reported a problem with *SetComment* under 1.3. It seems that Matt set a comment that was larger than the allotted size, and the file system copied the comment right on through the file name and subsequent fields of the file header block. This caused a rather spectacular crash, repeatable on each reboot until the offending file was removed. Getting rid of that file was not so easy. The file name was now destroyed, so its hash value was wrong, and the checksum was wrong. With the wrong hash value, it was impossible to delete the file. Matt used *DiskEd* to get rid of the file, but most Amiga owners are not so knowledgeable. He made a couple of suggestions for Commodore: (1) *SetComment* should not be permitted to write more than the allotted length, and (2) the Disk Validator should not only check for impossible filename lengths, but should also make sure the hash value and the filename match. This problem is not something we can correct ourselves, but we can work around it by watching the size of our comments while we wait for Commodore to repair it.

It was also Matt Dillon who reported problems with the 1.3 printer files in the Native Developers' Kit. He found that there is an XREF missing in each of the files, but Matt says that if you just remove that reference when the error message appears, the assembly can continue without it.

SPEAK:

Timothy Meisenheimer reported a problem with the SPEAK: device that has been confirmed as a bug, but one that is easy to remedy. The examples given show OPT to be in upper case, but it must be lower case in order to work properly. If only all of our solutions were so simple!

Resident

The most frequently reported 'bug' in 1.3 turns out to be a misunderstanding instead. It has been reported by many people and on all the boards that there is an error in the UseCount with *Resident*. No matter how many times you have used a command, it most often shows a UseCount of 0. It turns out that UseCount reports the number of tasks currently using a command, not the number that have used it since power-up. If no task is using it at the moment, the count will be 0.

Disks with gurus

Another problem that continues to cause trouble is disks that won't validate properly. This usually brings up a requester stating that the key is out of range, or that a key is already set, or that a key is already free. The key, of course, is the sector number of the file header. If it is already set, that sector is claimed by a different file for its own header, and you can't have two files with the same sector for the header. If the key is out of range, the file system has written in a sector number that is more than 1759 for a floppy, or larger than the highest sector number in the partition on a hard disk. These can be corrected by hand if you are good with a disk editor, but can usually be handled more easily by *DiskSalv*. The problem is

most often caused by a flaw in the disk media itself, but can be caused by a disruption in the power to the computer, or by something else that disturbs the drive during a write. If it happens to you often, you probably are using disks that are old and worn, were not of good quality to start with, or were single-sided. In the case of floppies, better disks is the best answer.

Another corrupt disk caused a most interesting crash on Jim Butterfield's Amiga. Jim had tried to replace a file on a disk that was too full. The save had appeared to be successful. When the disk was placed in the drive the next time, all went well until one particular icon was called. Fireworks erupted on the screen, pitiful cries came from the sound chips, and the machine went down. Jim knew that I like to study corrupt disks with strange problems, so he sent that one to me. Examination of the disk revealed that the icon had grown to some 37K in length. When the icon file was examined, the executable program was found to be attached to the end of the .info file. In this case, I copied the files over to another disk and saved the corrupt one for further study. DiskCopy will not work in a case like this - it would copy the errors as well as the good files. Copy takes a long time, but it takes only the good files over.

Hidden files?

Another interesting disk problem I have not yet solved. There was no guru, nor any other warning that all was not well. While preparing a monthly disk for my club, I had been adding icons to all of the files using one of the programs that adds icons automatically. There were drawer icons of several types and colours on the disk; I deleted all of them in order to replace them with drawers that matched. The program refused to make new icons for the directories. I then tried to copy a drawer icon over by hand, and that failed. Next, I called for a directory, and found no files. By this time, it appeared that the entire day's work was lost. In desperation, I used the correct path to call for a program, and it loaded and ran. Every program for which I could remember the correct name would run with no problems, but they would not appear in a directory or in a list. If I asked for a subdirectory whose name I could remember, it came up in the normal fashion. I solved the immediate problem by copying (not diskcopying) the files to a good disk, using 'Copy df0: to df1: all'. Every file copied correctly to the new disk, where it behaved in a normal manner. I still do not know what caused the original to lose all those files. I've looked at the original files with a disk editor several times, but have not yet spotted anything unusual that would cause this behavior. Perhaps one of you can explain this one.

Have you ever had a disk file that could not be deleted, renamed, or otherwise manipulated, yet it appears in your directory? You can see the file, right there every time you ask for the directory, but nothing you do accesses the file in any way. This is caused by having the pointer to the file in the wrong place. A file pointer (the number of the header block) must be placed in the directory slot that corresponds to the hash value of that file name. If it is in a different slot, the file system can still find the file name and put it in a directory, since it reads the sector number from the directory block, and then gets the name of the file from the header-block sector. Going backwards is a different matter. When you ask for a particular file, the system first gets the hash value of the file name, then looks in the slot corresponding to that value, and goes to the sector number it finds there to get the file. Suppose the hash value of a file name is 12 (it must be between 6 and 78, inclusive). If the header block were #898, then the directory entry on the disk would be something like this:

slot 0	|002|	the file type
slots 1-10	-----	
slot 11	|xxx|	location of file with hash value of 11
slot 12	|898|	location of header block for our file
slot 13	|xxx|	location of file with hash value of 13

After checking slot 12, the system reads sector 898, jumps down to slot 107 (which is byte 432) and reads the file name. If the number of the sector header is in the wrong slot, there will be false information in slot 12, so you will not be able to find the file. The number might be zero, indicating that no file by that name exists; it might be the number of a block in which another file begins, and the file names won't match. The entire hash chain will be checked and at the end, the system will give up, having found no file with the right name. In a case like this, if you can locate the actual file on the disk, you can get its header block number from the beginning of any block, get the hash value of the filename, and then place that block number in the slot corresponding to the file name. Most of the disk editors allow you to search the disk for a string. You can enter the file name into the string and the editor will find the header block for you.

RAD:

Many people are using RAD: happily, booting from it and using it as a Workbench in memory. Others find it to have many problems. Something in my own machine causes problems with any recoverable RAM disk. I find RAD: to be of little use because on every reboot it reports a read/write error, and just rebooting doesn't get rid of the error. It is more trouble than it is worth to me.

Happier users of RAD: have been asking how they can reboot from RAD: even though they forget to remove a floppy from the drive. Someone on Usenet (I think it was Andy Finkel) told how to do this. If you set the BootPri field in the mountlist entry for RAD: to 127, your machine will automatically boot from RAD: instead of from the floppy. Of course, you will find it impossible to boot from a floppy after you've done this, so you can't have it both ways. Well, impossible unless you first use *RemRad*.

DF1: for a 2000?

On the 2000, the internal drive is df0:, but an external floppy will be recognized as df2:. Many programs expect to find flop-

pies in df0: and df1:. They make no allowance for other configurations. This is not kind on the part of the programmers, but the user needs a way around the problem all the same. If the mountlist in the devs: directory has an entry that gives the name as df1:, but has a 2 in the unit field (UNIT = 2), then the user can add a line to the startup-sequence that says 'Mount df1:', and all references to df1: will go to device 2, getting the proper result. In other words, the external drive will now respond to either df1: or to df2:. It makes the software happy. In 1.3, the mountlist is already set up, and all you have to do is add the line to the startup-sequence. For those still using 1.2 (does anyone?), the '/*' and '*/' after the '#' sign must be removed, and 'UNIT = 2', and 'HIGHCYL = 79' must be aded.

Replacement drives for the sidecar

During the summer, I purchased a SideCar to go with my Amiga 1000. My need for IBM compatibility is extremely small, however, and the SideCar became a Christmas present for a very special relative. He was delighted, and spent a great deal of time getting things fixed to his liking - but on the third day, the drive stopped working. We tried to have it repaired, but were unsuccessful in the day or so that he would be here. We decided to purchase a replacement, but could not locate a Chinon drive to match the broken one. We bought a TEAC drive, and it wouldn't run. Knowledgeable friends came over to help, but nothing worked. The SideCar was fine - and the drive was fine, but they seemed unable to cooperate.

A solution was finally found, and it was very simple. There had to be a jumper between RY and ML. Once that was put in place, the drive functioned perfectly. So easy, once we had the knowledge.

Edited fonts

One final bug was reported on Usenet by Geoffrey Kim. He had edited the Topaz 8 font, saved it under a new name into his 'fonts:' directory, and then tried to use it with FastFonts. He received an error message stating that he could not use a proportional font - but his font was not proportional. This occurred because of a 1.2 bug in FED. FED was not clearing the ROMFONT bit in the flags field, and yet it was setting the DISKFONT flag. FF would not accept a font that had both flags set. Confused, maybe? With the 1.2 FED, you can use NewZap to correct this flaw, but it is easier to get 1.3: the new FED seems to have the problem corrected.

The Lazarus virus?

Several people (Amiga newcomers, no doubt) have recently reported a new virus both to CompuServe and to Usenet. They said that this virus attacked when they were trying to recover a disk, even though they were using the method approved by the system software. The disk became flawed, the requester came up telling them to use *DiskDoctor* to fix the disk, and they did. When they regained control of the computer, the disk had its structure completely rearranged, some of the files were missing, and the disk had been renamed to "Lazarus". This is not a virus, but is a characteristic of the *DiskDoctor* program.

That *Diskdoctor* would sometimes rename a disk as "Lazarus" became obvious to me back in 1985. *Diskdoctor* will also put almost all of the files into the root directory, and it makes other changes to the disk that prevent any further attempts at recovery. It is not a good thing to use, though I'm told that the 1.3 version is far more robust, and that it tells you more about what it is doing to your files. This quirk of renaming disks happens when *DiskDoctor* finds the root directory to have problems, especially if something prevents the reading of the disk name. In that case, the disk is given the name "Lazarus" to indicate that it is has been resurrected. Despite this resuscitation, the disk will almost certainly have serious problems. The files should immediately be copied to a good disk, and the disk named Lazarus should be reformatted - or tossed.

The IRQ virus

Another virus that is real has surfaced since the new year began. This one is the IRQ virus, and it may be the most troublesome to come thus far. This one is very real and very bad. It attaches itself to the first command in the startup sequence, installing itself instead of the load hunk, and putting the regular load hunk into the program itself. It manifests itself by changing the title of the first screen to its own name, "IRQ VIRUS". Steve Tibbett has already provided a killer for this virus, but it will be very difficult to keep it from infecting us over and over. It can attach itself to any program that runs, and it can be sent along in files uploaded or downloaded.

In addition to Steve's killer, may I suggest that readers get a copy of Jim Butterfield's SYSCHECK? SysCheck will read your WorkBench disk and report to you the name of every file that is not the same as the 1.3 release version. Well... almost every file. He doesn't check the startup-sequence, but does check all of the C commands, the libraries, the devices, and other files that users are unlikely to customize. By running SysCheck over your WorkBench regularly, you will spot the files that have been contaminated. They can then be replaced with the originals in order to clean your machine. The combination of SysCheck and IRQKiller is about the best protection we have as this goes to press.

* * *

A computer without problems or oddities would be far less interesting, at least to me, than one that has a few. Somehow, these peculiarities seem to give the machine a personality. New quirks are discovered constantly, and finding the solutions teaches us more about the machine. Have you noticed that people are still finding strange things in the Commodore 64, after all these years? How much more challenging it will be to search out all of the idiosyncrasies of the Amiga!

GFA-BASIC 3.0 for the Amiga

$139.95

GFA SYSTEMTECHNIK

Boldly goes where no BASIC has gone before.

- High-Speed Interpreter for easy program development
- Over 300 powerful commands
- FAST! – Execution times comparable to C
- In-line C and Assembler Commands
- Easy access to all Amiga libraries
- Extensive Amiga commands with submenus and built-in file requester
- Built-in Text Editor with syntax checking, procedure hiding and auto-indenting
- 400-page comprehensive manual
- Includes Run-Time Interpreter

Available NOW

from

Antic Software

544 Second Street, San Francisco, CA 94107

Call today to order:

800-234-7001

or see your local Amiga dealer.

(GF9200)

GFA-BASIC 3.0 is a trademark of GFA Systemtechnik, Germany; Antic Software and Antic are trademarks of Antic Publishing, Inc.; Amiga is a registered trademark of Commodore-Amiga Inc.

ARexx To You Too

For those who are asking, "What is ARexx?"

by Jay Gross

To hear some programmers tell it, ARexx is the third wonder of the known universe. Well, it is a nice thing for programmers, but it has implications for normal people, too. See, even though ARexx is a programming language (wait! don't go yet!), it offers benefits that regular computer users can thrill to, though they don't necessarily even have to know it's there.

The first thing to do is give an example. Sitting at your Amiga, you come across a long lost address and telephone number for your Great Aunt Martha. How do you get the information into your addressbook? The electronic addressbook, of course. Paper? Bah! Old tech!

Great Aunt Martha starts with an 'M', so you <M>ultitask out of whatever software you're running and crank up your database program. You get the addressbook application going and add the aunt. No problem. Re-index, close down, and you're back where you were, with one extra name in your Phonelist database.

Still, shouldn't the Amiga be doing all the work? That is, open up the database program, load the application, insert the name and information, save, resort, pack and index the file, close the application, close the database program, and announce the results of all this activity.

That's what ARexx is for. If the maker of the database software is thoughtful, you'll get an ARexx macro (a program by any other name is a macro) with which to accomplish that feat of magic, and you won't have to know the nitty gritty of how it works to make it work for you, just as you don't have to know how a C compiler works to take advantage of the database program that stores the addresses.

All this 'interprocess communication' is what ARexx is all about, and getting it all to work is the domain of programmers. *Using* it is for the rest of us.

An ARexx operating system?

With enough communication between processes, your Amiga could operate like one giant piece of software, even though in reality ARexx programs were making a bunch of different pieces of software do the work. The big thing is, too, that the pieces of software that so merrily interface with each other through ARexx need not all come from the same company. Neat.

Even if the software makers don't provide them (as they should), ARexx macros - née programs - aren't hard to do. If you can mangle something together in BASIC, you can program in ARexx. The syntax is remarkably similar (though sometimes infuriatingly different). ARexx talks to the system a lot more readily than BASIC does, too.

Okay, you BASIC enthusiasts out there, how much code would it take (count the 103,500-byte AmigaBasic interpreter) to get the directory of a disk. You have to get a DOS library going and do a SADD call to it. T'ain't easy. Consider the following ARexx sourcecode to get the directory of a disk:

```
/* Directory Getter Program */
address command
'dir'
exit
```

That's it, and ARexx's interpreter is only 32,756 bytes. You want to pass a command-line parameter on what directory to get? Okay, here goes...

```
/* Directory Getter Version 1.1 */
parse arg whereto
address command
'dir' whereto
exit
```

Easy. The point is, ARexx is a 'very high level language'. That means you get a whole lot of action for just a little bit of encouragement from the programmer.

ARexx's power and fame come mostly from its ability to set things in motion with Amiga commands (the 'address command' part of the code samples). So, if the AmigaDOS CLI can handle a command being sent by ARexx, why couldn't another Amiga program? Aha! The ARexx port!

The port side

Consider what you do while operating a program. You give it commands, either with key sequences or by clicking the mousebuttons. Suppose the programmer who wrote the application - your word processor for example - had installed a place for commands to be sent from the Amiga's operating system - a place agreed upon by ARexx and the program for messages to be posted. The application would then have another place to receive commands or data from, and it could do the same things for that 'port' as for the one you can see.

Aside from passing information back and forth, ARexx offers the ability to pass actual program *control* to an ARexx host. Programs can control other programs, or be controlled by others, or control the system, or honour commands passed from a CLI window. Imagine ARexx as an automatic, remotely controlled, programmable train that goes around to every depot in the system, getting stuff, dropping stuff off, bringing stuff back, and leaving orders for things to do at each stop.

When its 1.1 upgrade is shipped, Incognito's BBS software, Atredes, will have an ARexx port, a dandy way to give complete control of a system from a remote log-in. Remote terminals can be virtually any computer capable of modem telecommunications.

That means you could call up your home-bound Amiga from the Clone in your office and have the Amiga execute programs (through ARexx), sending program output to files for later retrieval, or back to you through the modem. Then if your secretary needs Great Aunt Martha's address, you could ring up and get it. An ARexx macro would open the database program, load the application, fetch the information, and send it back to Atredes, which would send it down the line to you.

If that doesn't truly boggle your mind, consider that one of Atredes' beta sites is Emerald Systems, makers of the Amiga Magellan program, an artificial intelligence applications generator. Magellan is getting a full ARexx port in its next update, and it already has some limited ARexx capabilities.

Think about it. A computer with artificial intelligence capabilities answers the phone (electronically). It can then take in information, make decisions or offer advice, and consult other programs (Microfiche Filer Plus, for example) for information in the process. Power!

The IBM 'standard'

ARexx was written by Bill Hawes - he wrote ConMan. It is based on REXX, which is a language 'standard' on mainframe computers and, for what it's worth, it's also IBM's 'standard'. REXX, was a procedures language originally developed by IBM for its mainframes to replace JCL. REXX stands for Restructured EXtended eXecutor language. What that means is that REXX is a way to write programs and manage other programs. Just as the Amiga Workbench provides a graphical interface to software, AREXX provides both a CLI interface and an interprocess communication interface to software.

The Amiga implementation adds a few extensions to the REXX language but, generally, a REXX program is an ARexx program, and vice versa. Of course, you can't issue the command **LIST df1:** on a mainframe and expect to get coherent results.

Lately, IBM is pushing their SAA (Systems Application Architecture), in the hope of setting 'standards' in everything from user interfaces to network protocols. ARexx is pretty much a superset of their 'Procedures Language', which is primarily based on REXX. How far SAA will affect hardware and software design in the industry remains to be seen, but REXX continues to grow in popularity, having been ported to many different machines large and small. Yes folks, there are REXX implementations on Clones. Yawn.

No compiling

If you missed the compile string up there in the ARexx source code department, there isn't one. ARexx is an interpreted language and, unless you're stacking loops several deep and manipulating huge strings, it cooks along at a *very* fast clip. The language recurses happily and, since it's operated out of an Amiga shared library (the interpreter), there can be many ARexx programs running at once in the system, all calling the same library at the same time. Isn't multitasking wonderful? ARexx programs can spawn or even generate other ARexx programs, which can then be run simultaneously. The possibilities here are indeed only as finite as your imagination and your Amiga's memory, not necessarily in that order.

Before ARexx can do something from within a program, support has to be added at the programmers' level. After that, the user can be left to his own devices for creating ARexx scripts to run (booo!), or ARexx programs (usually called macros) can be supplied with the product. From a user's point of view, the latter course is much better.

A growing number of Amiga software packages already have ARexx support, and many others are in the process of adding it; as update time comes around, you can look for more and more ARexx support in the future.

Alas, there aren't nearly enough pre-written macros and scripts for specific programs, except for some text editors and AmigaTeX, the typesetting program.

Everybody who writes about ARexx claims it's easy to use (see paragraph eight). To create simple scripts and such, yes, it is easy, although the documentation could be much better. Like any programming language, it does take some learning. There are many ways to do things, and it takes some experimentation (which is really kind of fun) to figure out what does what.

For non-programmers, ARexx's biggest confusions are when to use quotation marks and when capital letters are irrelevant or infuriatingly requisite. You don't have to worry much about appropriating reserved keywords as variable names, but if you get too sloppy, your code might be totally un-debuggable.

Variables? I do declare.

As a language, ARexx has some unusual features. For example, all variables begin life as character strings. You don't have to declare variables as integer, character, or whatever. (Those 'whatever' variables are really tricky to declare, too!).

If you put a number in a variable, the variable is numeric. You can even have the interpreter tell you what sort of stuff is stored in a variable, thus enabling your program to avoid taking the factorial of 'Great Aunt Martha'.

ARexx dynamically allocates and deallocates variables as needed. Any operation checks whether the operands contain valid material. The language provides operators for string concatenation, arithmetic, string and numeric comparison, and logical operations. You know about 'logical operators'. They let you take the Bool by the horns. AND, NOT, OR, XOR. Booleans of 'em!

As far as program statements, ARexx has plentiful flavours of DO. Things like DO WHILE, DO UNTIL and DO i = 1 to 10, meaning do something ten times. DO FOREVER, even, which takes a really long time to execute. BASIC programmers will cherish the SELECT statement, but C programmers will claim it's like SWITCH. The familiar IF/ELSE is there if you need it, else there's a SIGNAL statement useful for creating error traps.

GOTO is gone, too. Sorry, no GOTO in ARexx. Structured language programmers nag us novices to death about GOingTO, even though right there in the midst of the fabulously structured C language is a trusty ol' GOTO, waiting for the compiler to have a go at it to hide it from prying eyes.

ARexx gives you BREAK, ITERATE, and LEAVE, so you can get your program flow to flow around where you want it to go, without the eyebrow-raising GOTO.

A program doesn't do much worth worrying about if it doesn't have any of what programmer types call I/O. That means "input-output". There are various places on a computer that can put things into a program - the serial port, for modems; the keyboard for people. Besides that, there are several places for the computer to put out. The screen, so people can see it; the disk drives, to preserve it for posterity.

ARexx lets a program act as a host with its support library functions OPENPORT(), CLOSEPORT(), WAITPKT(), GETPKT(), and GETARG().

A wide variety of other built-in functions facilitates parsing strings - that's taking them apart into word pieces, like spelling things with alphabet soup - plus string manipulation, bit manipulation, memory allocation (if there is any) and so forth. The string manipulation facilities are particularly fast and powerful.

While you figure out the nitty gritty of using it as a programming language, you can use ARexx as purely a script language. ARexx scripts can be as powerful as you have the patience to make them.

Amiga programs which currently support ARexx include VLT, the VT100/Tektronix terminal emulator by Willy Langeveld; Charlie Heath's TxEd+ text editor (Microsmiths Software); AmigaTeX, the Amiga implementation of the TeX graphics description language, by Tom Rokicki of Radical Eye Software; Inovatronics' C.A.P.E. Macro Assembler; Cygnus Ed Professional from ASDG; and the unusual, graphics-based database, MicroFiche Filer Plus from Software Visions.

Recent newcomers to the ARexx support clan are Precision's Superbase Professional (version 3.0); Nag 3.0 from Gramma Software; Atalk-III telecommunications package from OXXI; and Magellan from Emerald Systems. Atredes version 1.1, previously mentioned, is in beta test, shipping Real Soon Now.

There is an ARexx version of MicroGNUemacs floating about, as well as SpeechToy. The source code for these programs, which is freely distributable, provides examples of how to implement the proper ARexx interface.

In addition, several companies have announced planned ARexx support for their product updating schedules, among them WordPerfect. There is even a screen-saving utility coming from Image Tech which can be told to operate from ARexx as easily as with the command keys.

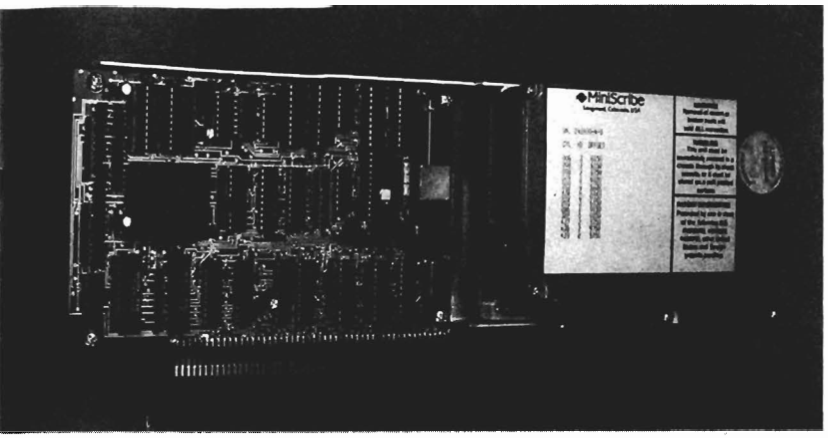
The key is user and programmer support

The key to ARexx, of course, is the support of the Amiga community - not just the users but software vendors and particularly programmers. It doesn't cost much, in money or time or code size, to add ARexx support to a software product, but without that support, ARexx is 'just' a nice macro language.

For users, ARexx has a lot to offer, even as 'just' a macro language. For programmers, it might 'just' become a necessity, once the users find out how useful to them ARexx support in application programs can be.

HardFrame/2000

The *Super-Speed*, DMA, SCSI Hard Disk Interface for the Amiga® 2000



- **AutoBoots AmigaDOS 1.3** ***(Price Includes HardFrame Eprom!)***
- ***Directly* Boots the New Fast-File System!** ***(Doesn't Need Old FS!)***
- **Auto-mounts All Hard Disk Partitions** ***(no Mount List Required!)***
- **Designed-in, Ultra Strong, Multitasking Performance**
- **High Quality Metal Frame for Stable, On-Card, Hard Disk Mounting**
- **Power Cabling Directly from Card to Disk**
- **50-pin Cable Included**
- **Supports up to *seven* SCSI hard disks of any size**

How fast is *fast*? **HardFrame/2000** transfers data at Amiga bus speeds! It's actually faster than the hard disk mechanism itself! And even more important in the Amiga's multitasking environment, **HardFrame/2000** has extremely efficient DMA circuitry to get on and off the bus in almost no time at all: 280ns to get on; 200ns to get off. And it's true, dedicated DMA, too! **HardFrame/2000** autoboots *and* automounts *directly* into the AmigaDOS™ 1.3 Fast File System (old file system partitions are *not* needed!). The core of any DMA SCSI interface is in its SCSI protocol chip and DMA chip. MicroBotics has chosen the new, high performance Adaptec AIC-6250 SCSI chip, capable of up to 5 megabytes per second raw transfer speed, and the Signetics 68430 DMA chip running at 12.5 megahertz. Then we added additional FIFO buffering and enabled 16-bit wide data transfers for maximum throughput. The sophisticated design of **HardFrame/2000** provides for automatic SCSI arbitration, selection and reselection. The hardware supports either synchronous or asynchronous data transfer. **HardFrame/2000** can function as either the SCSI bus initiator or the target and can reside in a multiple master environment. Physically, **HardFrame/2000** is optimally flexible: the compact, half-size card comes attached to a full length, plated aluminum frame. The frame has mounting holes positioned to accept standard, 3.5" SCSI hard disk units such as those manufactured by MiniScribe, Seagate, Rodime, and others (hard disk mechanisms must be supplied by the user or his dealer as a separate purchase item). Alternatively, you can cable-connect to a SCSI drive mounted in your Amiga's disk bay or in an external chassis. As many as seven hard disks may be connected to a single **HardFrame/2000**. There is no size limit on each disk. **HardFrame/2000** includes a 50-pin SCSI cable and header connectors for either 50-pin or 25-pin cable connection. Also included is a current tap to power frame-mounted drives directly from the slot itself. **HardFrame/2000** comes complete with driver, installation, and diagnostic software.

Available NOW! Suggested list price, $329 (hard disk not included)
Frameless version: $299.00. See your Amiga Dealer.

The HardFrame/2000 photo shows the product with a MiniScribe twenty megabyte hard disk installed. Hard disks are *not* included in the purchase price of HardFrame. Note that if placed in the first slot, HardFrame uses only one slot even with a disk attached.

New!

8-UP! (DIP) FastRAM

Another great memory board from MicroBotics, 8-UP! (DIP) is the "brother" of the original 8-UP! (which uses SIMMs and PopSIMMs to fill its memory space). 8-UP! (DIP) uses conventional 1 megabit RAM chips in standard sockets to provide your Amiga 2000 with 2, 4, 6, or 8 megabytes of autoconfiguring FastRAM! 8-UP! (DIP) is a super efficient CMOS design for lowpower consumption and high reliability. Suggested list price, $199 (0k installed)

Join MicroBotics
ONLINE TECHNICAL SUPPORT CONFERENCE ON BIX
(The Byte Information Exchange)
–call 1-800-227-2983
for BIX membership information!

MicroBotics, Inc.
Great Products Since the Amiga Was Born!
811 Alpha Drive, Suite 335, Richardson, Texas 75081 (214) 437-5330

Tell your dealer he can quick-order from MicroBotics directly – no minimum quantity –show him this ad!

"Amiga" is a registered trademark of Commodore-Amiga. "HardFrame/2000", "8-UP!", "PopSimm", are trademarks of MicroBotics, Inc.

Assembly Language Tricks

Three optimization levels for better machine code

by John Toebes
Copyright © 1989 John A. Toebes, VIII

John Toebes is a senior systems analyst at SAS Institute Inc. in Cary, North Carolina, and is the author of recent releases of the Lattice C compiler. He has been programming the Amiga since 1985, and is the coordinator for The Software Distillery, which is responsible for many popular freely redistributable programs including Hack and Hack Lite, PopCLI, MemWatch and Blink. He can be reached on Bix (JTOEBES) and the Software Distillery BBS (919-471-6436).

It is often said that if you really want to do something fast, you must do it in machine code. While there is some truth to this, it is far more important that, in whatever language code is written, it be written well. With assembly language particularly, it is possible to write bad, inefficient code without even realizing it. Only by learning to recognize these bad practices can one hope to learn to write better code.

In a very short time, an experienced assembly language programmer can look at a piece of code and categorize it as one of:

1) Great
2) Good
3) Anything else not good enough to be categorized

There aren't any hard and fast rules that one can use to make such a categorization, but there are some general guidelines. These guidelines tend to vary slightly from person to person; however, there is rarely disagreement as to when a particular piece of code qualifies as 'good' or 'great'. It is the intention of this article to help you write code that is recognized for its merits.

Great code tends to stand out quite easily - it tends to utilize almost every trick in the book (often inventing a few along the way) and is arranged so that the code 'happens' to fall through to the right areas. *Good* code tends to be easy to follow without any truly glaring mistakes or missed optimizations, but often loses out on organizational aspects. Bad code, on the other hand, tends to be, well... not so good.

We can divide the strategy of writing optimal code into three basic areas:

1) Peephole or instruction optimizations
2) Local optimizations
3) Global optimizations

In fact, if these terms seem similar to those used in compiler technology, it is because they are. Compilers have to go through a similar process in making decisions about generating code, although they often don't have as much information available to them as the assembly language programmer.

Peephole optimizations

In the area of actual instruction selection, the 68000 family offers many opportunities for alternative instructions with its wide range of address modes and instructions. We will focus here on the 68000 itself, as it is the most common processor, and these optimizations do apply to the later processors. Some guidelines that we can follow in choosing instructions are:

1) Avoid immediate mode 32-bit values
2) Avoid 32-bit operands with address registers
3) Avoid 32-bit absolute address modes
4) Avoid the MUL and DIV instructions
5) Use the QUICK form of instructions when possible
6) Forget that the CLR instruction exists

Let's try applying these rules with a few simple instructions:

```
L1:   move.l  #1,D0          6 bytes    12 cycles

L2:   add.l   #2,A0          6 bytes    16 cycles

L3:   move.l  #3,Datavar    10 bytes    28 cycles

L4:   muls.w  #10,D0         4 bytes    70 cycles

L5:   clr.l   D0             2 bytes     6 cycles
```

These would more effectively be coded as:

```
L1:   moveq   #1,D0            2 bytes     4 cycles

L2:   addq.w  #2,A0            2 bytes     8 cycles

L3a:  moveq   #3,D0            2 bytes     4 cycles
      move.l  D0,Datavar       6 bytes    20 cycles
                              --------   ---------
                               8 bytes    24 cycles

L3b:  moveq   #3,D0            2 bytes     4 cycles
      move.l  D0,Datavar(A4)   4 bytes    16 cycles
                              --------   ---------
                               6 bytes    20 cycles

L4:   move.l  D0,D1            2 bytes     4 cycles
      lsl.l   #2,D1            2 bytes    12 cycles
      add.l   D1,D0            2 bytes     8 cycles
      add.l   D0,D0            2 bytes     8 cycles
                              --------   ---------
                               8 bytes    32 cycles

L5:   moveq   #0,D0            2 bytes     4 cycles
```

As you can see from these simple instructions, there is not always an even trade-off. In the L4 example, we doubled the code size but were able to cut the execution time in half. Since the code increase is so small, this is a particular situation where it is reasonable to decide that the time savings outweighs the increase in size.

With the L3 example, we have an excellent example of a case where generating two instructions actually saves both in code size and in execution time.

This brings up an important additional rule:

7) If you can use a MOVEQ for a 32-bit constant - *do it!*

This applies to any instruction where you can use a scratch data register. For example:

```
and.l  #15,D2     6 bytes  16 cycles
or.l   #2,D3      6 bytes  16 cycles
sub.l  #28,D4     6 bytes  16 cycles
cmp.l  #1,D1      6 bytes  14 cycles
                 --------  ---------
                 24 bytes  62 cycles
```

Would come out much faster and shorter with:

```
moveq  #15,D0    2 bytes   4 cycles
and.l  D0,D2     2 bytes   8 cycles
moveq  #2,D0     2 bytes   4 cycles
or.l   D0,D3     2 bytes   8 cycles
moveq  #28,D0    2 bytes   4 cycles
sub.l  D0,D4     2 bytes   8 cycles
moveq  #1,D0     2 bytes   4 cycles
cmp.l  D0,D1     2 bytes   8 cycles
                --------  ---------
                16 bytes  48 cycles
```

But wait - you don't always have to use a register. Sometimes you want to subtract a small number from a register but the number is too big to use for the SUBQ instruction. For this, it is reasonable to use two SUBQ instructions in a row. Thus:

```
sub.l  #10,D0     6 bytes  16 cycles
```

becomes:

```
subq.l #8,D0      2 bytes   8 cycles
subq.l #2,D0      2 bytes   8 cycles
                 --------  ---------
                 4 bytes  16 cycles
```

Another common area where immediate constants play a role is in stack manipulation. When calling a C routine that requires parameters on the stack, cleaning up from the call generally looks like:

```
add.l  #12,SP     6 bytes  16 cycles
```

To add eight or fewer bytes, you should instead use:

```
addq.w #8,SP      2 bytes   8 cycles
```

For nine or more bytes, you can use the LEA instruction

```
lea    12(SP),SP  4 bytes   8 cycles
```

Lastly, you should not limit yourself to simple MOVEQ instructions to load constants into registers. Depending upon the constant, there are a number of tricks that you can use to load in a value. For example, to load the value $10000 into a register you might be tempted to do this:

```
move.l #$10000,D0  6 bytes  12 cycles
```

However, a sharp programmer can instead do:

```
moveq  #1,D0      2 bytes   4 cycles
swap   D0         2 bytes   4 cycles
                 --------  ---------
                 4 bytes   8 cycles
```

Clearly this works for any number in the range $FF800000 to $007F0000 where the low order word is all zeroes. Another trick of this sort applies when loading a constant that is just a little too big to load with a MOVEQ instruction:

```
moveq  #200-256,D0   2 bytes   4 cycles
neg.b  D0            2 bytes   4 cycles
                    --------  ---------
                     4 bytes   8 cycles
```

In fact, any 8-bit value can be loaded into a data register in this form with the MOVEQ/NEG.B. For an even cleverer trick (though admittedly less frequently encountered), you can follow a MOVEQ with a NEG.W instruction to generate constants of the form $FFFF00xx and $0000FFxx.

If you're after small code, you can load up single bits at the expense of a little speed with the ROL and ROR instructions. To generate $8000000 to $01000000 you would use:

```
moveq  #1,D0        2 bytes   4 cycles
ror.l  #n,D0        2 bytes   8+2n cycles
```

To set a single bit in the range from $00000080 to $00004000 you can use:

```
moveq  #$40,D0      2 bytes   4 cycles
rol.l  #n,D0        2 bytes   8+2n cycles
```

Given these tricks, in four bytes you can actually generate almost all of the commonly encountered constants for which you would normally have needed six bytes. This does add up in the long term.

These optimizations have all applied to data registers; here are some tricks that apply to address registers:

8) Use the **.W** form of instructions for address registers when possible
9) Do operations in address registers if you can

For example, if you needed to load a constant value into an address register, you might have coded:

```
move.l #502,A1      6 bytes  12 cycles
```

Instead, you can trick the LEA instruction into doing your work for you:

```
lea    502.w,a1     4 bytes   8 cycles
```

Of course, if you wanted to load a zero into an address register, you could do even better with:

```
sub.l  a1,a1        2 bytes   8 cycles
```

The **.W** form of the instruction is even more useful for pushing constants onto the stack in preparation for a call to a C routine. Instead of the expected:

```
move.l #1,-(SP)     6 bytes  20 cycles
```

or as we know a little better to do:

```
moveq  #1,D0        2 bytes   4 cycles
move.l D0,-(SP)     2 bytes  12 cycles
                   --------  ---------
                    4 bytes  16 cycles
```

We can instead use the PEA instruction:

```
pea    1.w          4 bytes  16 cycles
```

Though for small constants that MOVEQ can handle this doesn't seem to be a savings, it does save for constants that fit in 16 bits but are too big for the MOVEQ. On the 68020 and the 68030, the MOVEQ sequence is actually faster for those constants that it can handle, due to pipelining in the processors, so you are probably better off using it when you can.

On the Amiga, one address register manipulation that occurs frequently is converting a BPTR into an APTR. Since a BPTR is shifted right 2 bits, one might be tempted to use a data register to perform the operation (since shift and rotate instructions do not operate on the address registers).

Even transferring to a data register is expensive:

```
exg    A0,D0        2 bytes   6 cycles
lsl.l  #2,D0        2 bytes  12 cycles
exg    D0,A0        2 bytes   6 cycles
                   --------  ---------
                    6 bytes  24 cycles
```

Using a MOVE instead of an EXG instruction only saves us four cycles. We can do much better than that by instead doing two instructions:

```
add.l  A0,A0        2 bytes   8 cycles
add.l  A0,A0        2 bytes   8 cycles
                   --------  ---------
                    4 bytes  16 cycles
```

Certainly there are many other peephole optimizations available on the 68000. However, these are the most prevalent and useful for typical code. An excellent method of learning them is to sit down with a good Motorola manual and read the timings. Some of them are actually quite surprising.

However, these optimizations by themselves are only a small part of what makes an assembly language good (or *great*, as we would like it to be). In reality, you can get much more of a gain by making algorithmic changes to your code than by making simple instruction substitutions.

For this reason, we now examine the range of local optimizations available.

Local optimizations

Unlike the peephole type of optimizations, local optimizations require a little more thought in order to arrange, and often have minor effects on the algorithm you are trying to implement. The simplest one that comes to mind is in operations that would correspond to a multiply in a high level language. For example, to index into an array of items, it is necessary to scale the index by the size of the operand. In assembly, a good

programmer deals with sizes that are powers of two. For example, instead of:

```
move.l    index,D0
muls.w    #4,D0
```

you would code:

```
move.l    index,D0
lsl.l     #2,D0
```

On the subject of shifts, there are some simple rules to follow:

10a) If the number to shift is greater than 15, do a swap instruction, then subtract 16 from the number to shift
10b) If the number is 8 or more, generate a shift by a constant 8 and then subtract 8 from the number to shift
10c) If the number is 1 and the shift is left, generate an add instruction instead.
10d) If the number is greater than 0, generate a shift instruction for that amount.

Taking these rules, we get the following sequences:

```
Shift D0 left by  1:    add.l D0,D0
Shift D0 left by  2:    lsl.l #2,D0
Shift D0 left by  4:    lsl.l #4,D0
Shift D0 left by  8:    lsl.l #8,D0
Shift D0 left by  9:    lsl.l #8,D0
                        add.l D0,D0
Shift D0 left by 10:    lsl.l #8,D0
                        lsl.l #2,D0
Shift D0 left by 16:    swap  D0
                        clr.w D0
Shift D0 left by 24:    swap  D0
                        clr.w D0
                        lsl.l #8,D0
Shift D0 left by 25:    swap  D0
                        clr.w D0
                        lsl.l #8,D0
                        add.l D0,D0
Shift D0 right by  1:   lsr.l #1,D0 (no optimization)
Shift D0 right by 16:   clr.w D0
                        swap  D0
Shift D0 right by 25:   clr.w D0
                        swap  D0
                        lsr.l #8,D0
                        lsr.l #1,D0
```

All of these sequences turn out to be better for the 68000 than loading a shift constant into a register and doing the shift, as they use one less register and are all faster. This is particularly true for the 9-15 range. When generating the 16 case, always remember to do the swap in the correct place. It is important to note that these shifts really aren't very careful with the sign bit, but in assembly programming this is rarely important. If you do have to handle a signed shift of 16 or more, you can use the EXT.L instruction instead of CLR.W.

One particularly fancy optimization that is useful for anyone writing an assembler *in* assembler takes advantage of forcing bits out as the shift is being done. In generating the register target field of an instruction, it is necessary to mask the value with a 7 and then shift it left 9. A good assembler programmer might use this code (assuming the value to be already in D0):

```
moveq     #7,D1        2 bytes  4 cycles
and.l     D1,D0        2 bytes  8 cycles
lsl.l     #8,D0        2 bytes 24 cycles
add.l     D0,D0        2 bytes  8 cycles
                       -------- ---------
                       8 bytes 44 cycles
```

But a more astute programmer would have noticed an opportunity here (knowing that the register value fit into a byte), and coded instead:

```
lsl.b     #5,D0        2 bytes 16 cycles
lsl.l     #4,D0        2 bytes 16 cycles
                       -------- ---------
                       4 bytes 32 cycles
```

11) Track condition codes with a vengeance

A great assembly program almost never has a TST instruction in it. The 68000 is very good about setting condition codes to reflect almost every operation. Only when you are putting something into an address register is the condition code left unchanged.

This can bite you, however, if you simply add code to an existing program without checking condition codes, as almost anything can destroy condition codes. At times it can be more beneficial to remember the MOVEM instruction which, although it is larger and slower than the MOVE instruction, can safely load or store a single register without affecting the condition codes.

It is a good rule of thumb to forget the TST instruction (if possible) and arrange your code so that the condition codes fall through.

Condition code tracking is extremely useful for multi-way branching. Assembly language programmers are afforded the simple luxury of testing for three things at once:

```
add   val1,D0      ; do some operation to set the CC
                   ;  (not a TST)
blt.s negative     ; if negative we need to do some
                   ;  more
beq.s done         ; at zero we are complete
;  otherwise we come through with a positive value
```

In most high level languages (with the exception of Fortran) this type of situation is almost never seen, but in good assembly programs, values and flags are arranged so that this situation can be used.

When choosing the branch instructions, it is important to be knowledgeable about the frequency of the code paths to be taken. If something is likely to occur, then the branch condition should be such that you know that it will *not* be taken the majority of the time.

Error situation detection should be done by a test that branches out of line, allowing the normal case to continue past the branch instruction. In this way, there is no pipeline break on the higher processors, and even on the 68000 you take advantage of the shorter cycle time for not taking the branch.

```
        moveq   #1,D0
        and.l   D0,D1     ; is the error request flag set?
        bne.s   error1    ; yes so go tell them about the
                          ;  error
;  normal fall through here in the most common case
```

12) Abuse the DBcc instruction

The best instructions of all, however, are those of the DBcc series. Even on the 68010, these put the processor into a mode in which the instruction covered by the loop is kept in a cache for fast repeated execution. However, the DBRA instruction has some very important drawbacks to be aware of:

12a) The loop counter is only 16 bits. The high order portion of the register is untouched.
12b) For performance you should only include a single instruction in the loop.
12c) Loops entered at the top are *zero*-based, executing one more time than the number in the register.

Even with these limitations, the DBRA instruction can be used in many situations. The most common is in copying areas of memory:

```
        moveq  #15,D0     ; copy of 16 bytes of memory on
                          ; any boundary
        move.l source,A0
        move.l dest,A1
lab:    move.b (A0)+,(A1)+
        dbra   D0,lab
```

If you don't know the length ahead of time, this is easily remedied by entering the loop at the DBRA:

```
        move.w len,D0
        move.l source,A0
        move.l dest,A1
        bra.s  dbr
lab:    move.b (A0)+,(A1)+
dbr:    dbra   D0,lab
```

However, the DBcc instructions are much too powerful to be left to copying bytes. They are great for comparing memory, searching for characters, and even filling memory. The trick is just to use the condition codes to terminate the loop.

For example, if you wanted to search for a character in a string, you could do this:

```
        move.w len,D0
        move.l string,A0
        moveq  #char,D1    ;character to search for
        bra.s  dbr
lab:    cmp.b  (A0)+,D1
dbr:    dbne.s D0,lab
```

Now in this case we had to be careful. The length could have been zero, so we need to start at the bottom of the loop. However, the loop would have terminated if the condition code had been EQ, so we make sure that the last thing we do before starting the loop actually sets the CC to a non-EQ value - hence the MOVEQ before the BRA.

Global optimizations

These peephole and local optimizations allow you to fine tune hot spots in the code and in general make the microscopic cases run faster. However, for the most return on effort, there are several global optimizations that are a win:

13) Take the time to choose registers carefully

A good program attempts to assign registers to the high traffic items, or even to the most important at a particular instant. The secret to a great program is arranging the code and register choices so the value needed is always in the right register. In particular, it is desirable to avoid having to do a MOVE register to register instruction just to get something in place for a call.

14) Avoid MOVEMs when possible.

Because the standard 68000 calling conventions require you to preserve all registers except D0/D1/A0/A1, it is necessary to save any other destroyed registers on entry and restore them on exit. It is most desirable to make register selections so that as few registers as possible are saved. In fact, if you only need to save a single register, substitute a MOVE instruction instead.

15) Avoid the LINK and UNLK instructions.

While the LINK instruction is frequently used in high level languages, its use in assembly programs is actually quite rare; it is only useful for creating temporary buffers/storage for items too large to fit in a register. If there are not enough registers to go around for a particular operation, requiring storing some on the stack, it is sometimes better to create a subroutine for a portion of that operation (if time is not critical), or even push the parameters on the stack instead of storing at a fixed location in an allocated frame.

16) Make intelligent choices for flags and values

If you have three possible states for a flag then choose -1, 0, and 1, so that you can use a three-way branch. If the value will

be used to index a table, choose even ones that correspond directly to the entry index, in order to avoid a multiply. For bits, put the most commonly tested bit into the sign position so that the single bit gets set as a side effect of loading the flags. With a long, you have 3 bits that can set the sign bit based on the word length - pick these first.

17) Use tables when appropriate

Many operations can be done readily with tables. For example if you are doing a transformation for on-screen rotations, it may be sufficient to precalculate the sine for all 360 degrees, then round off the rotation angle desired to the nearest degree and look it up directly, instead of going through an expensive calculation. Jump tables to perform operations (especially when the number for the operation is intelligently selected as above) are particularly quick.

The heavy artillery

By no means is this a complete list - the opportunities for optimization on the 68000 are endless. However, the amount of time to *perform* the optimizations is finite. For this reason, it is necessary to weigh the importance of a particular optimization against the time necessary to implement it. One excellent tool to consider using is a profiler. This will point out where in the code the most time is being spent, and allow you to concentrate on that one area. A good rule of thumb is that 95% of the time will be spent in about 5% of the code.

There is little benefit to optimizing the remaining 95% if you haven't paid particular attention to the hot 5%.

Parting trick

18) Learn how to use the X bit.

The X bit is one of the strangest bits in all the 68000 architecture. It is set by very few instructions, but does have the property of allowing you to set an entire long word to 0 or -1 based on its value:

```
subx.l  D0,D0
```

With this in mind, test your ability to optimize 68000 code by figuring out the best sequence of instructions that will do the following: when given a value in D0, with condition codes in an unknown state, set D1 to 0 if D0 currently contains a 0, and to a 1 otherwise. You may destroy D0 if necessary. The answer is given upside-down in the box just below - but give it a try yourself before you peek!

The Answer!

```
neg.l   D0
subx.l  D1,D1
neg.l   D1
```


1 MB/min HD BU rate achieved!

ExpressCopy Only $44.95 US

"The Fast, Safe, and Reliable road for HardDisk Backup and Archival."

ExpressCopy can perform hard disk backups and copy files from hard disk to floppy disk at a rate of up to 1MB per minute, 600k Bytes per minute with verify of backup disks. Backup disks have the same directory structure and file attributes of the original files, and can be used normally. Either Normal or Fast FileSystem compatible disks can be written. Files to be copied can be selected by their DateStamp, filename pattern matching, the Archive Bit, and by source directory. Options for setting the Archive bit, verifying the data written to floppy disk, and estimating the number of disks required for the backup. Up to 4 copies of the backup disks can be created at a time, or disks can be pre-loaded in up to 4 disk drives. New disks are automatically formatted and verified. If a bad disk is found during verify, you can insert a different disk and restart the backup from the beginning of that disk. No need to completely restart the backup. Parameters to be used for backups can be saved in configuration files.

Designed for effective multitasking. Change ExpressCopy's priority while it's running. You can HALT the backup and use the Disk Drives normally, then RESUME the backup.

Backup restoration can be done using any file copy program, your favorite Directory Utility, or by ExpressCopy's Restore program.

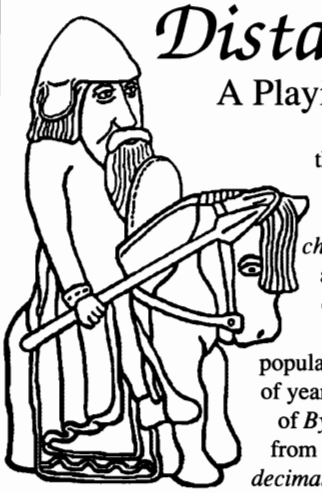
ExpressCopy has no copy protection, and can be used from both the CLI and Workbench.

Express-Way Software, Inc.
PO Box 10290, Columbia, MO 65205-4005
(314) 474-2984

Workbench is a trademark of Commodore-Amiga,Inc.

Distant Armies

A Playing History of Chess



Distant Armies is a journey through distant times and distant places. Survey the evolution of chess since the invention of *chaturanga* over a thousand years ago in India. Play exotic games of chess from Burma and China as well as intermediate forms popular in Islam and Europe hundreds of years ago. Play on the round board of *Byzantine chess* or defend attacks from powerful combination pieces in *decimal chess*. Features include 2 and 3 dimensional views, several levels of difficulty, a mode to show all of the legal moves for a piece, and extensive online descriptions of the rules and history of each game in the set.

Play and learn ten historic forms of chess:
Chaturanga, Shatranj, Burmese, Chinese, Byzantine, Mediaeval, Courier, Turkish, Decimal, and Los Alamos.

Eagle Tree Software
P.O. Box 164
Hopewell, VA 23860 U.S.A.
(804) 452-0623

Distant Armies is
$44.95
for Amiga with 512K

Amiga is a registered trademark of Commodore-Amiga, Inc.

Shell Scripts Under 1.3

More power, more options, more kludges than ever!

by Andy Finkel
Copyright © 1988 Andy Finkel

Andy Finkel is the Manager of Amiga Software at Commodore. He has been involved with Commodore machines at Commodore since before the introduction of the VIC 20.

Writing scripts is an ancient and dishonourable form of programming. Everyone does it, but somehow a well-written script never obtains the respect that a well-written C program does. Some people even deny that scripts are programs. Yet scripts continue to be written and used. (In fact, many features found on some versions of Unix™ began as script files, later translated into C.)

In my opinion, script writing is certainly programming. However, it is a strange sort of programming, where the power and even syntax of each keyword (program) varies wildly. When you write scripts, programming style seems to go out the window... getting the job done is the primary goal, and the most incredible kludges are used as a matter of course.

I've always written scripts, on every computer I've used that had even a partial script language. In some ways it's more fun than programming in a real computer language. With scripts you are continually faced with the challenge of making commands do more or work differently than the author intended. You just don't get that feeling when working with C (where the closest equivalent is to see how few parentheses you can get away with using).

One of the improvements in Workbench 1.3 was some upgrading of the commands to make them more useful in scripts. (Another was the SCRIPT bit, which tells the Shell to Execute a file, rather than Run it).

Some of the things that were not easy to do in a script are now possible. Others are now easy. Still others require brand new incredible kludges.

To demonstrate some of the new features, I've included a few scripts with this article. They are:

catargs a utility script that combines a number of arguments into a single string

nextarg a utility script that strips off the first argument from a set of arguments, passing the original argument back

print.spooler an upgrade of the script-based 1.2 print spooler. This one's cleaner, and can be modified to handle graphic files.

talking.clock yes, that's right - a script-based talking clock, using no specially-written commands. Every multitasking system needs a talking clock, if only as an impressive demo to show the power of multitasking. (Okay, okay, I'm kidding. Still, it is a strange thing to use!)

Before I get to the individual scripts, it's probably a good idea to review some of changes that can help scripts, And maybe sneak in a little discussion of features that have long been present, but have been underused... In fact, let's start with those.

The **?** operator of AmigaDOS does much more than display a cryptic template and request user input - it is a truly amazing and useful feature. (Okay, so I get carried away.) There are two abilities of the **?** I want to draw your attention to: the **?** prompts only for *missing* arguments, and the input prompt can be redirected.

Why is this so powerful? Well, let's say in a script you want to copy a file to DF1:Tools. And you want the user to be able to specify what file is to be copied. The easy way to do this is:

```
echo "What file please ?"
copy >NIL: to df1:tools ?
```

That's all there is to it. The output redirection to NIL: sends the template to never-never land, to make the script look better. This is also a good place to use the new NOLINE option of the Echo command... but I'm getting ahead of myself.

I mentioned earlier that the input stream for this function can be redirected. This lets a command take *command line input* from other commands. For instance, you could set the date of a file to the current date (forgetting for the moment that the SetDate command will do this anyway) with the following:

```
date >ram:qwe
setdate <ram:qwe >NIL: filename ?
```

Again, the ability to fill in only *missing* arguments comes in handy... the filename is specified on the command line itself. This even works with scripts. You can say:

```
execute <ram:qwe scriptname File ?
```

and it will work as if scriptname was a command.

Okay, now onto 1.3, and some of the new features.

The first new feature is that your scripts will work slightly differently because Execute can finally write to the T: directory. Under 1.2, Execute always wanted to write to :T. That was often your Workbench disk, which, if you know what's good for you, was always write-protected. But now Execute can finally create the temporary files it always wanted to, because we have T:, which usually points to RAM:T. So what is different?

Well, first of all, your scripts will be able to unwind properly. When one script calls another, upon the second script finishing, script execution will continue with the remainder of the first script. Second, when one script calls another, variable substitution will take place. This has its good and bad points. The bad point is that redirection in the second file may be taken as an improper attempt to use script variables, so Execute will complain "No .K Directive". The good point is that variables will work in the second script. To avoid the bad point, just get in the habit of putting the lines:

```
.bra {
.ket }
```

near the top of your scripts. These lines change the variable delimiters to { and }, which are much nicer to look at anyway.

If you want to have scripts work the same as in 1.2 (and not use the T:) just cancel the T: assignment. Execute will once again use :T.

The other good points about using T: for scripts are based on the way scripts currently work. This probably isn't a big surprise to most of you, but Execute scripts are implemented in a stream fashion. Execute manages an input stream; *Skips* advance through this input stream by characters. The proper unwinding of Script files (where one script file calls another and, when the second one finishes, the first one picks up where it left off) and the script variable substitution are implemented by copying a portion of the script to the T directory. (Under 1.3, this is assigned to RAM:T in the standard Startup-Sequence).

This change lets scripts get faster, and lets you write-protect your Workbench disk but still get variable substitution.

The second important new feature is the presence of the IconX program. This program allows execution of scripts from Workbench. If your script starts with a '.' (like a .key or .bra statement), IconX will treat your script as a full script, with variables, branching, I/O, etc. If your script doesn't require any of this, leave the '.' out.

Now, a quick listing of the major changes. We'll get into how to use them in a minute.

Assign Exists

The Assign command can now be used to quietly check if a particular device, assignment, or volume exists and is mounted. The command executes without accessing the device/assigmment/volume in any way.

List with the LFORMAT option

LFORMAT lets you use the List command as a script generator, by making it possible to format output of a List QUICK in any manner you want. For instance, this List command:

```
list >ram:qwe #? lformat="protect *"%s%s*" -d"
```

will generate a file containing a Protect command for each file in the current directory. If you now execute this file, you change the protection of each file in the current directory, even though the Protect command doesn't have wildcards. The SPAT and DPAT scripts found on the 1.3 Workbench disk are one way to use this capability.

Echo

Using the NOLINE option, the Echo command will not output a linefeed automatically. This lets you format your script output more neatly, and lets you create files without a closing linefeed. When you are building scripts from other scripts, this capability can be critical. Using the FIRST and LEN keywords, the Echo command can also be used to parse simple strings, especially those stored in environment or shell variables.

To gain those additional features, the Echo command was forced to pick up a template. This is a bit disconcerting to those who were used to using the **Echo ?** command to pause scripts - now an ugly template shows up instead of the **:**. Well, to those of you I say: use the ASK command. That's what it's for, after all.

Endskip

Endskip is used as a marker by the Skip command. EndSkip acts as a barrier that tells Skip to continue execution on the following line. In 1.2, if the Skip command was told to skip to a non-existent label, it would go to the end of the file, and the script would terminate.

With Endskip, the Skip command can now be used to get input from the user (numbers, words) and branch to the appropriate part of the script. This allows you to implement menus in your scripts, for instance.

Eval

Eval is a simple command - it evaluates simple expressions, and it does conversions. More powerful versions are easy to write (I have one here you'll see in a future release), but it does the job. You can use it to calculate loop variables, as a hex to decimal conversion program, etc.

{$$} in scripts

Actually the { and } should be your current variable delimiters, set by the .bra and .ket statements. But since I recommend changing the defaults to { and }, that's how I'll list them here. Anyway, the {$$} gets replaced with the current CLI/Shell number during script execution. This can be used to create unique temporary file names - a critical feature in a multitasking system, where you may be running the same script more than once at a time. By appending the CLI/Shell number onto a filename, and always doing so, you make sure you have unique temporary files.

ENV:

ENV: is supposed to be used to store some of your run-time preferences: the name of your favourite editor for use in MORE, things like that. But since there are commands to set and get those variables, it is something that is immediately usable in strange ways by shell programmers. Under 1.3, ENV: is just an assign. Later, it will probably be made into an independent handler. For our purposes, they are identical.

If

If now has expanded comparisons: greater than (GT) and greater or equal to (GE). When you combine these with the original equal (EQ) and not (NOT), you can now specify GT, GE, EQ, NE, LT and LE. The VAL keyword makes the comparison a numeric one; otherwise it is a string comparison.

PIPE:

The PIPE: device is useful when taking the output of one command and feeding it to another. Because it blocks when empty, you can use it to make sure that the command that needs input stays in sync with the command providing output.

Search

The Search command now has useful returns, so you can branch off the results of a search. It also has some useful optional output modes.

Shell

The aliases the Shell provides make script execution look nicer, as does the SCRIPT bit. The Resident command, when used with the Shell, will let you write scripts that don't need to access the C: directory, and take up less memory than when those commands are stored in a RAM disk. Finally, the >> redirection operator will allow you to append data onto files, instead of replacing a file. This lets you easily build script files programatically (especially when used with Echo NOLINE).

Now for the scripts themselves...

Menus and choices

One of the more interesting script functions you could do under 1.2 was implementing menu selections using the Skip command. There were two problems, however. One of these was just cosmetic: the prompt was always on a separate line from the question mark. The other was more serious: if the user typed an illegal selection, your script would terminate. And there wasn't anything you could do about it. Why? Because the Skip command had read the entire input stream, that's why.

The solutions

The first problem is solved by the new NOLINE option of the Echo command. Echo does not do a CR/LF after outputting the string when this option is used. The second problem is solved by the use of the EndSkip command. As explained earlier, EndSkip acts as a 'fence' for Skip. When Skip finds one of these in a file, it stops skipping, so execution continues at the command immediately following the EndSkip statement.

Example

```
;-----------------------------------------------------
lab start
Echo "1. Instructions*N2. Format Hard Disk*N3. Calculate
     the last digit of PI*N"
Echo "*NPlease indicate your choice " noline
skip >NIL: ?
;
lab 1
lab help
lab instructions
Echo "Sorry, no help available.*N"
skip start back
;
lab 2
lab format
Echo "Be Serious.  In a Demo ?*N"
```

```
skip start back
;
lab 3
lab calculate
Echo "This isn't an episode of Star Trek, you know.*N"
skip start back
;
Endskip
echo "I'm sorry, I can't do that."
skip start back
;-----------------------------------------------------
```

The second example shows how to implement a script that executes a specified number of times. This script combines Eval and environment variables (accessed through the If command) to count down; like the previous example, it also uses the Skip BACK command to jump back in the file.

```
;The Loop example

.key loop/a
; demo a loop using eval and skip
.bra {
.ket }
echo >env:loop {loop}
lab start
echo "Loop #" noline
type env:loop
eval <env:loop >NIL: to=t:qwe{$$} value2=1 op=- ?
type >env:loop t:qwe{$$}
IF val $loop GT 0
skip start back
endif
echo "done"
;-----------------------------------------------------
```

Next are two utility scripts that handle argument processing for scripts. Here you'll see a very interesting use of one script calling another. The call to the second script allows arguments to be modified in a number of ways. The *catargs* script takes up to ten arguments, an optional prefix string, and an optional postfix string, and places all of those arguments into a file, with no spaces between them. You can then use the file contents as a single argument to a standard AmigaDOS command.

```
; The catargs utility
.key n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,
     file/k,pre/k,post/k,noline/s
.bra {
.ket }
.def file *
echo >{file} "{pre}{n1}{n2}{n3}{n4}{n5}{n6}
     {n7}{n8}{n9}{n10}{post}" {noline}
;-----------------------------------------------------
```

The *nextarg* utility takes the arguments passed it, and puts all but the first argument into a file. This allows scripts to skip arguments.

```
;The nextarg utility
.key n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,
     file/k,pre/k,post/k,arg1/k,noline/s
.bra {
.ket }
.def file *
echo >{file} "{pre} {n2} {n3} {n4} {n5} {n6} {n7} {n8}
     {n9} {n10} {post}" {noline}
IF {arg1} NOT EQ ""
echo >{arg1} "{n1}"
ENDIF
;-----------------------------------------------------
```

Next is an update of a script-based print spooler I did under 1.2. The 1.3 changes allowed me to make it 'cleaner' (though I hesitate to apply that concept to shell scripts). Anyway, the spooler is in two parts... the first is *Start-Spooler*, which initializes the *Spooler* for quiet background operation, by making the commands Resident, and copying the scripts into RAM:.

The second part is the *Spooler* itself, which just waits quietly for something to appear in the spool directory, then copies it to PRT:. The spooler only works with text files. I leave it as an exercise for the reader to convert it to spool graphics files as well (Hint: use the Cmd command) and to make it output the date and a nice title line.

```
. Start-Spooler :set up things for print spooler
failat 21
resident c:type ;resident things for speed
resident c:lab  ;this is a replace operation - it's fine
resident c:wait ;if they are already resident
resident c:skip
resident c:run
resident c:delete
assign >NIL: spool: exists ; spool already assigned ?
IF warn                    ;no, better make one
IF not exists ram:Spool
makedir ram:spool
makedir ram:spool/t
endif
assign spool: ram:spool
copy sys:Empty.info ram:Spool.info
     ;so spool can be used from WB
copy s:Spooler spool:t
endif
;
run >NIL: execute spool:t/spooler

.key "" ;shell script print spooler, V2.0
; SPOOLER The print spooler itself
lab start
list >spool:t/dospool<$$> spool: files
     lformat="type %S%S to prt:*Ndelete %S%S"
run >NIL: execute spool:t/dospool<$$>
wait 3 mins
skip start back
;-----------------------------------------------------
```

And now - the talking clock!

Finally, the extremely useful and interesting talking clock, implemented using standard commands in a shell script. The talking clock uses the *catargs* and *nextargs* scripts above.

When running this script, I've always found it helpful to place all three scripts into RAM:, as well as making all the commands used Resident. This lets it run quietly in the backgound, speaking only at the proper time. There are too many tricks in this one to mention them all.

By the way, I didn't feel the 1.3 AmigaDOS commands were finished until this particular application was possible. Remember that famous axiom of computer science: "If a talking clock can be programmed in a language, then anything can be programmed in that language."

```
;
;The talking clock
;
.key mode,alarm,message
.bra {
.ket }

; mode sets whether to speak the hour only, or speak
     every 15 minutes
; alarm is an optional time to say a message, and
     message is the message

.default message ring,ring,ring

date >t:{$$}tim   ;get just the time into tim
execute <t:{$$}tim >NIL: nextarg file = t:{$$}qwe ?
execute <t:{$$}qwe >NIL: nextarg file = t:{$$}tim ?

; now get the hour into hour, and the minutes
     into minute
execute <t:{$$}tim >NIL: substr file=env:{$$}hour
     f1=1 l1=2 file1=env:{$$}minute f2=4 l2=2 ?

; now set the alarm time (must be on the correct
     interval)
echo >env:{$$}a.hr "{alarm}" first=1 len=2
echo >env:{$$}a.min "{alarm}" first=4 len=2

; build the string we say
eval <env:{$$}hour >NIL: op=mod
     value2=12 to=env:{$$}hours ?
;
lab loop
;
IF val ${$$}minute GE 45
echo >env:{$$}minute "00"
echo >t:{$$}minutes " o clock"

eval <env:{$$}hour >NIL: 1 + to=t:{$$}tim
     lformat="until %N:00*N" ?
eval <env:{$$}hour >NIL: 1 + to=t:{$$}qwe ?
eval <t:{$$}qwe >NIL: to=env:{$$}hour ?
eval <env:{$$}hour >NIL: op=mod
     value2=12 to=env:{$$}hours ?

IF val ${$$}hour GE 24
echo >env:{$$}hour "00"
echo >env:{$$}hours "12"
echo >t:{$$}tim "until 00:00"
ENDIF
skip loopend
ENDIF
;
IF val ${$$}minute GE 30
echo >env:{$$}minute "45"
echo >t:{$$}minutes " forty five"

eval <env:{$$}hour >NIL: to=t:{$$}tim
     lformat="until %N:45*N" ?
skip loopend
ENDIF
;
IF val ${$$}minute GE 15
echo >env:{$$}minute "30"
echo >t:{$$}minutes " thirty"
eval <env:{$$}hour >NIL: to=t:{$$}tim
     lformat="until %N:30*N" ?
skip loopend
ENDIF
;
echo >env:{$$}minute "15"
echo >t:{$$}minutes " fifteen"
eval <env:{$$}hour >NIL: to=t:{$$}tim
     lformat="until %N:15*N" ?
;
lab loopend
wait <t:{$$}tim >NIL: ?

IF {mode} EQ "hour"
IF val ${$$}minute NOT EQ 0
skip acheck
ENDIF
ENDIF

echo >speak: "It is now "

IF val ${$$}hours EQ 0
echo >speak: "twelve"
ENDIF
IF val ${$$}hours EQ 1
echo >speak: "one"
ENDIF
IF val ${$$}hours EQ 2
echo >speak: "two"
ENDIF
IF val ${$$}hours EQ 3
echo >speak: "three"
ENDIF
```

```
IF val ${$$}hours EQ 4
echo >speak: "four"
ENDIF
IF val ${$$}hours EQ 5
echo >speak: "five"
ENDIF
IF val ${$$}hours EQ 6
echo >speak: "six"
ENDIF
IF val ${$$}hours EQ 7
echo >speak: "seven"
ENDIF
IF val ${$$}hours EQ 8
echo >speak: "eight"
ENDIF
IF val ${$$}hours EQ 9
echo >speak: "nine"
ENDIF
IF val ${$$}hours EQ 10
echo >speak: "ten"
ENDIF
IF val ${$$}hours EQ 11
echo >speak: "elaven"
ENDIF
IF val ${$$}hours EQ 12
echo >speak: "twelve"
ENDIF
type t:{$$}minutes to speak:

IF val ${$$}hour GT 11
echo >speak: "pe em"
ELSE
echo >speak: "ay em"
ENDIF

lab acheck
; do alarm message
IF {alarm} EQ ""
skip loop back
ENDIF

IF val ${$$}hour EQ ${$$}a.hr
IF val ${$$}minute EQ ${$$}a.min
echo >speak: "{message}"
ENDIF
ENDIF

skip loop back
endskip
execute talk.clock {mode} {alarm} {message}
;-----------------------------------------------------
```

These scripts should give you a start on writing your own. Remember, the kludgier you get, the more interesting your scripts become. Sure, you can (and probably should) use a scripting language like ARexx, which is a real programming language. But that kind of takes the fun out of forcing commands to do what you want, rather than what they want.

RAWCOPY V1.3

- Ultra-powerful backup tool for your AMIGA™
- Copies software that no others will.
- Deprotects many of your favorite programs.
- Supports up to four 3.5" disk drives.
- Mouse-driven user interface.

START | ABORT

SOURCE DRIVE: DF0: DF1: DF2: DF3:

TARGET DRIVES: DF0: DF1: DF2: DF3:

Read Variables
Drive 0 | Read Length=$3000
Cylinder 00 | Sync Word=$4489
Side Bottom Top | Sync On Word ON OFF

Write Variables
Drive 1 | Write Length=$3000
Cylinder 00 | Pre-Write=$0000
Side Bottom Top | Precompensation 00ns 140ns 280ns 560ns

READ TRACK | WRITE TRACK

Format Control MFM GCR | Bit Cell 2us 4us

See Your Dealer, or write or call for more information. Available on disk for $59.95. Enclose correct amount plus $2.00 for shipping, and specify UPS or first class mail delivery. Visa and MasterCard Charges accepted.

Micro Systems International™
1143 Monroe St.
Carleton, MI 48117
(313) 654-8402

Amiga Genealogy?

Now Shipping Your Family Tree

THE FIRST GENEALOGY DATABASE FOR THE AMIGA

- Standard Reports • Free Form Memo Area
- Supports Over 10 Million Family Members
- Easy To Learn and Use

YOUR FAMILY TREE

Contact Your Local AMIGA Dealer or Send $49.95 + 6% Sales Tax (PA. Residents Only). $4.00 Shipping and Handling (Within U.S.A.) to:
MICROMASTER, INC.
1289 Brodhead Road, Monaca, PA 15061
PH. (412) 775-3000

AMIGA is a Reg. Trademark of COMMODORE AMIGA

Using the Bison/Yacc Parser Generator

Part 1 of a series

by Eric Giguère
Copyright © 1989 Eric Giguère

Eric Giguère has been involved with the design of an ANSI-conformant C compiler at the Computer Systems Group at the University of Waterloo, where he is a computer science student in the co-op program.

Computer science has progressed to the point where program development can be aided and enhanced through the use of special tools and libraries. Compiler development is no exception to this. This set of articles is intended to be an introduction to compiler development and a primer on how to use one of the compiler tools available on the Amiga - the Bison/Yacc parser generator. We'll end our discussion in the next issue by using Bison to construct a non-trivial (and I hope useful) program based on compiler technology.

This article assumes no knowledge on your part of how a compiler works beyond what you may have learned by using one.

Translators, compilers and interpreters

A *translator* is a program that takes a list or script of instructions and converts them into a machine-executable form. Translators are usually classifed either as compilers or as interpreters. A *compiler* produces stand-alone code: once the instructions are translated, the compiler is not needed to execute them. An *interpreter* translates and executes the instructions simultaneously without producing stand-alone code. On the Amiga, programming in C is done with a compiler (Manx or Lattice) and in BASIC with an interpreter (AmigaBasic).

The distinction between compilers and interpreters has as much to do with trade-offs in translation time versus execution speed as with methodology. A compiled program will execute much faster than its interpreted version because it is pure machine code, but interpreters have a quicker program development cycle because the program is translated and compiled incrementally instead of in batches. Both types of translators share much in common, however, and so even though the rest of this article will talk about compilers, much of what is said will apply to interpreters as well.

Another type of translator is an *assembler*, which converts low-level assembly language into its pure machine language equivalent. Assemblers are really just specialized compilers - interpretive assemblers are only of use for debugging or teaching purposes.

The parts of a compiler

Conceptually, a compiler can be divided into four parts: scanner, parser, symbol table and code generator.

The *scanner* reads and transforms the source file into *tokens*. Tokens are the smallest components of a program that still have some meaning attached to them. Typical tokens include keywords, strings, identifiers (variable names), operators and numbers. Though they're not particularly hard to write, it's important for scanners to be efficient because they perform a lot of I/O. Scanning is also referred to as *lexical analysis*.

The *symbol table* holds all the information needed during program compilation: identifier names, scope levels, type definitions, and so on. It can be as simple as a linked list or as complex as a set of hash tables.

The *code generator* is the largest and most complex piece of the compiler, responsible for translating the stream of tokens provided by the scanner into machine code. The code generator is usually composed of several parts itself - the Manx C compiler code generator, for example, generates assembly language and then uses an assembler to generate machine code. One common scheme is to have a compiler generate *machine-independent* code, which a second compiler then translates into code for a specific machine. This separation of the compiler into front and back ends aids portability and allows for the quick development of other compilers.

The *parser* controls the compiler. It analyzes the stream of tokens the scanner sends it and decides which routines in the code generator to call. The parser is also known as the *syntactic analyzer*. It's important to note the difference between *syn-*

tax - the order and structure of tokens, and *semantics* - their meaning. A syntactically correct program isn't necessarily semantically correct - the code generator usually includes a *semantic analyzer* to determine if the code it is told to generate actually means anything. For example, the C expression **++ptr++** is syntactically correct, but the rules of C state that only one of the increment operators can be applied to the variable - the code generator will flag this as an error.

The types of parsers

Parsers come in two general classes: table-driven and recursive descent.

A *recursive descent* parser is one in which the syntactic rules of the language are described using recursive procedures. They 'descend' down the rules until one is found that handles the current token stream. A short example illustrating recursive-descent routines is found on page 51 of Volume 1, Issue 2 of *Transactor for the Amiga* in Chris Gray's article on the Draco compiler.

The recursive-descent technique is generally used in hand-coded parsers or when extensive error checking (and possibly error recovery) is required. It won't be discussed here.

Table-driven parsers use a set of generated tables to control the compiler. The tables tell the *parser driver* which sequences of tokens are valid and which routines to activate when it recognizes such a sequence. The same parser driver can be used in many different compilers - only the tables need to be changed. The exact methods a table-driven parser uses aren't important in our discussion. This type of parser is usually machine-generated by a program called a *parser generator.*

Yacc and Bison

Yacc is a parser generator developed at AT&T Bell Laboratories in the late 1970s. *Yacc* is an acronym for *Yet Another Compiler-Compiler*, since that is in effect what a parser generator is. Yacc is a standard part of the Unix development system and has gained great popularity among Unix hackers - compiler courses at universities are often taught on Unix machines using Yacc and C. The only problem with this is that Yacc can only be used on machines licensed by AT&T.

In the early part of this decade, however, Bob Corbett and others developed a Yacc lookalike, which they named *Bison* (if you don't get it, look up 'yak' in the dictionary). Bison is a freeware parser generator that takes the same input as Yacc and generates similar tables. Bison has been ported to many machines - an Amiga version is available on Fish Disk #136. (It should be noted that Bison is not in the public domain and is only intended for non-commercial or freeware software development.)

From now on, the term 'Bison' will be used to refer to both the Yacc and Bison parser generators.

Language grammars

To generate a parser for a specific language Bison must be given a set of syntax rules - a *grammar.* The grammar describes both the allowable token sequences and the *actions* (routines) to perform when valid sequences are recognized. A Bison grammar also lists other information the parser generator needs, but we'll get into specifics next issue. For now let's consider the Bison grammar in a more general sense.

A grammar is composed of terminal symbols, nonterminal symbols and metacharacters. *Terminal symbol* is another name for token. A *nonterminal symbol* represents a specific sequence of terminal and nonterminal symbols. Each *rule* of a grammar defines the sequence that makes up a nonterminal. *Metacharacters* are operators such as ':' and '|' that are used to define the grammar rules.

These concepts are best demonstrated by example. Consider the C language. A constant in C can be an integer, a floating-point number, a single character or an enumeration identifier. In a grammar for C we would code:

constant : T_INTEGER;
constant : T_FLOAT;
constant : T_CHAR;
constant : T_ENUM;

Here 'constant' is a nonterminal, T_INTEGER, T_FLOAT, T_CHAR, and T_ENUM are terminals, while ':' and ';' are metacharacters. Each grammar rule is of the form:

LHS : RHS ;

where LHS is a single nonterminal and RHS is one or more terminals/nonterminals. In effect, such a rule says that "the nonterminal labelled LHS is a synonym for the sequence labelled RHS", a kind of assignment expression. The ':' metacharacter separates the two sides of the assignment and the ';' ends the assignment.

(Important note: don't confuse metacharacters with tokens. For example, the semicolon above is used to end a grammar rule, *not* to indicate that a semicolon is part of the syntax described by the assignment.)

There is no obvious distinction between terminals and nonterminals in the grammar, only that each nonterminal must appear at least once as the left-hand side of a grammar rule. It helps, however, to visually distinguish the two in some way, as is done by using italic type for the nonterminals in the examples here. The names used for terminals and nonterminals must be unique and should describe the corresponding entity in some way, such as 'constant', 'expression' and 'statement'.

Returning to our example, we see that four different rules define what a constant can be. These rules can also be combined into one with the same effect:

constant : T_INTEGER
| T_FLOAT
| T_CHAR
| T_ENUM
;

Here we introduce a new metacharacter '|' for alternation. Thus 'constant' can be one of T_INTEGER or T_FLOAT or T_CHAR or T_ENUM. The semicolon is only used at the end of the last choice. Now consider this additional set of rules:

add_or_minus : *constant*
| *add_or_minus* T_PLUS *constant*
| *add_or_minus* T_MINUS *constant*
;

where T_PLUS and T_MINUS are the tokens for the plus and minus signs, respectively.

Suppose the scanner encounters the following input:

3 + 5.0 - 6

After tokenization it becomes:

T_INTEGER T_PLUS T_FLOAT T_MINUS T_INTEGER

For now we won't worry about how the actual values for each number are stored - consider only their tokens. When the scanner passes these on to the parser, it will start to apply the rules as described by the grammar and reduce the sequence of terminals as much as it can. First it replaces 'T_INTEGER' and 'T_FLOAT' with '*constant*':

constant T_PLUS *constant* T_MINUS *constant*

There is no rule to replace "*constant* T_PLUS" so the parser replaces "*constant*" using the rule for "*add_or_minus*":

add_or_minus T_PLUS *constant* T_MINUS *constant*

Now "*add_or_minus* T_PLUS *constant*" can be reduced using one of the alternate rules for "*add_or_minus*":

add_or_minus T_MINUS *constant*

And again:

add_or_minus

The original sequence of five terminals has now been reduced to a single nonterminal. The parser stops because the grammar has no other rules.

The goal of the parser is to use the rules of the grammar to reduce the token stream for the source program to a single nonterminal, the goal symbol. When it reaches this symbol, the parser is done and the compiler stops.

It's important to make sure that none of your grammar rules conflict: if two different rules use the same sequence of symbols on the right-hand side, the parser won't know which to choose. Such a grammar is called an ambiguous grammar. Though Bison can be made to choose between conflicting rules, it's best to avoid ambiguity altogether.

Actions and syntax errors

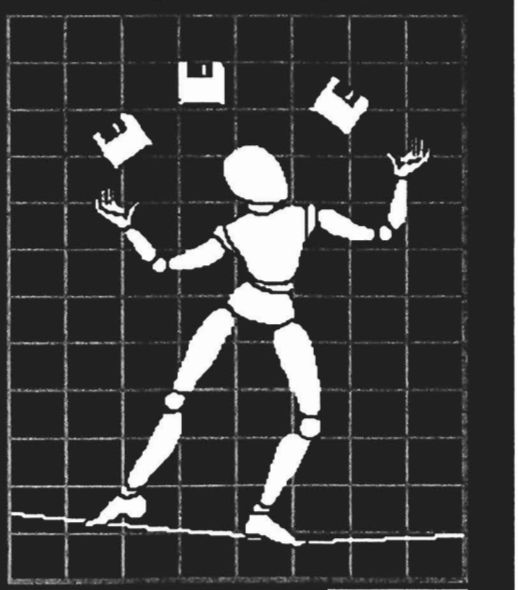
If all the parser did was replace terminals with nonterminals, it wouldn't be of much use except as a syntax checker. Each rule, however, may have a set of *actions* associated with it; these use the information provided by the parser to generate code. The actions are performed whenever a reduction (a replacement) occurs. Actions will be discussed in detail next issue. A syntax error occurs when the parser cannot find a rule to reduce the current input stream.

Next issue

So much for the theory. Next issue we'll delve into some specifics on using Bison to generate parsers and how to use those parsers in your own programs. We'll create our own grammar, scanner and action routines - it should be interesting! In the meantime, take a peek at Appendix A in Kernighan and Ritchie's *The C Programming Language* to see what the C grammar looks like in detail.

NEW Release
ver 1.10
- Full Intuition based Interface
- Selective Restore
- Amiga wildcards
- Multiple indexing for greater **safety**
- Over 907K per floppy

saf-T-net



Hard Disk Backup
for your Amiga
available at your dealers **now**
or from (714) 955-2555
Metadigm, Inc. 19762 MacArthur Blvd. Suite 300; Irvine CA 92715

AMIKIT IS HERE!

AMIKIT IS HERE!

No more frustrated hours spent staring at your computer screen! No more blundering through the Workbench or CLI -- the ultimate aid for novices and experts alike has arrived!

Step by step, AmiKit helps you explore and benefit from the awesome power of your Amiga. From the most basic to complex commands, AmiKit teaches you how to gain control and confidence using your machine, whether its an A500, 1000 or 2000. And what's more, AmiKit includes over thirty Public Domain software programs, including: Workbench 1.3; a virus checker; a database; a spreadsheet; a word processor; games; and much more!

AmiKit is one of the most effective ways sucessfully to navigate your way throughout the workbench and CLI.
As a special introductory offer from Transactor for the Amiga, you can get AmiKit for $34.95* (plus $2.50 for shipping) -- $5.00 off the recommended retail price -- an amazing bargain! When Amiga User International, the leading European magazine, reviewed AmiKit, they were extremely pleased with the package. We know you will be, too!

To: Transactor For the Amiga
10-85 West Wilmot St.
Richmond Hill, Ontario, CANADA
L4B 1K7
I would like to purchase AmiKit at the special price of $34.95 + $2.50 shipping ($37.45)*.
Please make cheques payable to Croftward Publishing, Inc., or debit my Mastercard/Visa account:

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiry Date _____

Signature _____

Name _____

Address _____

_____

_____

Telephone _____

*ONTARIO RESIDENTS ADD 8%

T9046 *Please allow 28 days for delivery*

Photocopies of this coupon accepted

Programming in M2Amiga

String things

by Anthony Bryant

Anthony Bryant is a computer engineer working with remote control systems and robotics. He programs the Amiga and other Commodore computers in his spare time.

Structured Strings

There are several methods of manipulating strings in M2. Since a string is simply an ARRAY OF type CHAR, you can use the structured approach to pass open ARRAY OF CHARs as procedure parameters and manipulate individual CHARs with M2's standard relational operators and the standard functions CHR(), ORD() and HIGH(). (M2Amiga's standard libraries contain about half a dozen string manipulating functions). I have included a potpourri of home-made functions (see m2strings.mod) to illustrate this method.

Note that HIGH() tells us the maximum number of chars the array can hold, but we must also check for shorter strings by the presence of 0C, the null-terminator. M2Amiga's compiler includes a built-in copy function, used to fill string variables; so that if we declare:

```
VAR
mystring: ARRAY [0..some length] OF CHAR;
```

...then the statement:

```
mystring:="only part of a string";
```

...fills the variable, mystring, with the string literal, 'only part of a string' and, if shorter than the declared length, sticks in a 0C null-terminator. M2Amiga stores all string literals in the constant space of the program, so in reality the above statement is equivalent to:

```
CONST
fixedstring = "only part of a string";
(* stored with null-terminator *)
. . .
mystring:=fixedstring;
(* calls the built-in copy function *)
```

Some restrictions of M2 are evident in my home-made functions (trying to imitate string functions from languages like BASIC and SNOBOL). For one thing, function results are limited to 8 bytes, so you can not have an ARRAY OF CHAR as a function result. Another is that M2 does not support string concatenation using '+', so there is a separate Concat() function.

These routines can serve as building blocks for even fancier string functions.

If you know for certain that all of your strings will be null-terminated you can eliminate the code check using the HIGH() function in order to gain a little extra speed and compactness, but you sacrifice universality.

Unstructured strings

As in C, you can also manipulate strings as a group of bytes (terminated with a null, of course) and accessed by an address which is the start of the string. The use of unstructured strings is mandated, for example, when interfacing with the Amiga system routines like those in the Intuition library.

Sample functions (see cstrings.mod) illustrate this unstructured approach. Note that rather than representing the strings using type ADDRESS, which in M2Amiga is defined as a POINTER TO BYTE, I have defined a new type STRING, defined as POINTER TO CHAR. This avoids having to do a CAST() back and forth between BYTE and CHAR, which you must otherwise do, since you cannot use relational operators on type BYTE (it's an undefined type).

If you mix the two methods, using arrays and addresses, you can do a type transfer between two objects that occupy the same storage space, BYTE and CHAR, as follows:

```
char:= CAST(CHAR,address^);
address^:= CAST(BYTE,char);
```

Using a combination of methods, you should be able to solve most programming problems in strings.

Note that the listings 'm2strings.mod' and 'cstrings.mod' are not really program modules (they lack a MODULE heading and END). One could, of course, make a library out of them (with a DEFINITION and IMPLEMENTATION part) but I preferred to just incorporate them as needed into my working source and modify them to suit.

The program 'BaseConv' is a little example of interfacing with strings at the system level - it's an easy to use decimal/hex converter that runs in a small window with two string gadgets. BaseConv.mod really is a program MODULE, here. I have used m2strings function UCase() and modified it for hex and null-terminated strings. ValToStr() and StrToVal() are functions from a standard M2Amiga library.

Additional note: The letter 'C' appended to an integer, like 0C - the null terminator - indicates the character is identified in ASCII code. For some unfathomable reason, in M2 this is given in octal! So, for example, carriage return is 15C, delete is 177C. Fortunately, M2Amiga provides a standard library of ASCII codes and you can always use the string literals, like " " for space and "" for a null string (the null terminator).

```
(* --------------- C - string manipulation functions --------------- *)

(* SAMPLE FUNCTIONS:   strlen(), strcat(), strncat(), strcmp(),
 *                     strcpy(), strrchr(), toupper()
 *)

(* NOTE BENE:   All of your strings MUST be null-terminated!!
 *              It is your responsibility to check the integrity of this
 *)

TYPE
 STRING = POINTER TO CHAR; (* rather than use an ADDRESS type *)

(* strlen() - return length of string (not including nul)
 *)
PROCEDURE strlen(text: STRING): CARDINAL;
 CONST  nul=0C;
 VAR  scan: STRING;
      count: CARDINAL;
 BEGIN
   scan:= text; count:=0;
   WHILE (scan^ # nul) DO INC(scan); INC(count); END;
   RETURN(count);
 END strlen;

(* strcat() - append string srctext to dsttext
 *)
PROCEDURE strcat(dsttext, srctext: STRING);
 CONST  nul=0C;
 VAR  dscan,sscan: STRING;
 BEGIN
   dscan:= dsttext;
   WHILE (dscan^ # nul) DO INC(dscan); END;
   sscan:= srctext;
   WHILE (sscan^ # nul) DO dscan^:= sscan^; INC(dscan); INC(sscan); END;
   dscan^:= nul;  (* terminate with nul *)
 END strcat;

(* strncat() - append at most n characters of string srctext to dsttext
 *)
PROCEDURE strncat(dsttext, srctext: STRING; n: CARDINAL);
 CONST  nul=0C;
 VAR   dscan,sscan: STRING;
       count: CARDINAL;
 BEGIN
   dscan:= dsttext;
   WHILE (dscan^ # nul) DO INC(dscan); END;
   sscan:= srctext;
   count:= n;
   WHILE (sscan^ # nul) AND (count > 0) DO
     dscan^:= sscan^; INC(dscan); INC(sscan);
     DEC(count);
   END;
   dscan^:= nul;  (* terminate with nul *)
 END strncat;

(* strcmp() - compare string text1 to text2 ;
 *            returns -1 for <, 0 for =, +1 for >
 *)
PROCEDURE strcmp(text1, text2: STRING): INTEGER;
 CONST  nul=0C;
 VAR  scan1,scan2: STRING;
 BEGIN
   scan1:= text1; scan2:= text2;
   WHILE (scan1^ # nul) AND (scan1^ = scan2^) DO INC(scan1); INC(scan2); END;
   IF (scan1^ = nul) AND (scan2^ = nul) THEN RETURN (0);
   ELSIF (scan1^ = nul) THEN RETURN (-1);
   ELSIF (scan2^ = nul) THEN RETURN (+1);
   END;
 END strcmp;

(* strcpy() - copy string srctext to dsttext
 *)
PROCEDURE strcpy(dsttext, srctext: STRING);
 CONST  nul=0C;
 VAR  dscan,sscan: STRING;
 BEGIN
   dscan:= dsttext;
   sscan:= srctext;
   WHILE (sscan^ # nul) DO dscan^:= sscan^; INC(dscan); INC(sscan); END;
   dscan^:= nul;  (* terminate with nul *)
 END strcpy;

(* strrchr() - find last occurrence of a character in a string
 *)
PROCEDURE strrchr(text: STRING; charwanted: CHAR): STRING;
 CONST  nul=0C;
 VAR  scan,place: STRING;
 BEGIN
   place:= NIL; scan:= text;
   WHILE (scan^ # nul) DO
    IF (scan^ = charwanted) THEN place:= scan; END; (* found char *)
    INC(scan);
   END;
   RETURN (place); (* found char, or NIL if none *)
 END strrchr;

(* toupper() - convert a string to all upper case
*)
PROCEDURE toupper(text: STRING);
 CONST  nul=0C;
 VAR   scan: STRING;
 BEGIN
   scan:= text;
   WHILE (scan^ # nul) DO
     scan^:= CAP(scan^); INC(scan);  (* convert character *)
   END;
 END toupper;
```

```
(* ---------------- M2 - string manipulation functions ------------------- *)

(* FUNCTIONS:  Len(), InStr(), LeftStr(), RightStr(), MidStr(), Concat(),
 *             Compare(), Asc(), String(), Spaces(), UCase(), LCase(),
 *             Trim(), Strip(), Justify(), Delete(), Remove(),
 *             DecStr(), HexStr(), BinStr(), StrVal()
 *)

(* NOTE BENE: Strings may be "packed" or "null-terminated".
 *            Call-by-reference, VAR type ARRAYs, are used for efficiency.
 *            Where strings are modified or created,  make sure the
 *            ARRAYS are sufficiently long enough to hold the new CHARs!
 *            If not enough space,  new string WILL be truncated!
 *)

TYPE
 LENSTR = [0..32767];  (* range of string length *)

(* Len() - returns the number of actual characters in string; 0 = nul string.
 *)
PROCEDURE Len(VAR string: ARRAY OF CHAR): LENSTR;
 CONST nul=0C;
 VAR   i,l: LENSTR;
 BEGIN
    i:=0; l:=HIGH(string);
    WHILE (i <= l) AND (string[i] # nul) DO INC(i); END; (* end-of-string *)
    RETURN (i);
 END Len;

(* InStr() - returns position of the first occurrence of string2 in string1
 *           starting at [n]th character
 *)
PROCEDURE InStr(VAR string1, string2: ARRAY OF CHAR; n: LENSTR): LENSTR;
 CONST nul=0C;
 VAR i,j,l,k,p: LENSTR;
 BEGIN
    l:=HIGH(string1); k:=HIGH(string2); p:= MAX(LENSTR); i:=0;
    WHILE (i <= l) AND (string1[i] # nul) AND (i < n) DO INC(i); END;
    IF (i = n) THEN
       REPEAT
         j:=0; p:= MAX(LENSTR); (* assume failure *)
         WHILE (i <= l) AND (string1[i] # string2[j]) DO INC(i); END;
         IF (i <= l) THEN
           p:= i;  (* remember position *)
           WHILE (i <= l) AND (j <= k) AND (string1[i] = string2[j]) DO
             INC(i); INC(j);
           END;
           IF (j <= k) AND (string2[j] # nul) THEN i:= p+1; END;
         END;
       UNTIL (i > l) OR (j > k) OR (string2[j] = nul);
    END;
    RETURN (p); (* if p = MAX(LENSTR) then occurrence not found *)
 END InStr;

(* LeftStr() - creates a string2 containing the leftmost [n] chars of string1
 *)
PROCEDURE LeftStr(VAR string1, string2: ARRAY OF CHAR; n: LENSTR);
 CONST nul=0C;
 VAR i,l,k: LENSTR;
 BEGIN
    i:=0; l:= HIGH(string1); k:= HIGH(string2);
    WHILE (i <= l) AND (i <= k) AND (string1[i] # nul) AND (i < n) DO
      string2[i]:= string1[i]; INC(i);
    END;
    IF (i <= k) THEN string2[i]:= nul; END;  (* terminate with nul *)
 END LeftStr;

(* RightStr() - creates a string2 containing the rightmost [n] chars of string1
 *)
PROCEDURE RightStr(VAR string1, string2: ARRAY OF CHAR; n: LENSTR);
 CONST nul=0C;
 VAR i,j,l,k: LENSTR;
 BEGIN
    l:= HIGH(string1); k:= HIGH(string2);
    i:= Len(string1); (* locate end-of-string *)
    j:= 0;
    IF (n > 0) THEN
      WHILE (j <= k) AND (string1[i-n] # nul) DO
        string2[j]:= string1[i-n]; INC(j); INC(i);
      END;
    END;
    IF (j <= k) THEN string2[j]:= nul; END; (* terminate with nul *)
 END RightStr;

(* MidStr() - creates a string2 starting at [n]th char of length [m] of string1
 *)
PROCEDURE MidStr(VAR string1, string2: ARRAY OF CHAR; n,m: LENSTR);
 CONST nul=0C;
 VAR i,j,l,k,p: LENSTR;
 BEGIN
    i:=0; j:=0; l:= HIGH(string1); k:= HIGH(string2);
    p:= m;  IF (p = 0) THEN p:= MAX(LENSTR); END;
    WHILE (i <= l) AND (string1[i] # nul) AND (i < n) DO INC(i); END;
    IF (i = n) THEN
      WHILE (i <= l) AND (j <= k) AND (string1[i] # nul) AND (p > 0) DO
        string2[j]:= string1[i]; INC(j); INC(i); DEC(p);
      END;
    END;
    IF (j <= k) THEN string2[j]:= nul; END; (* terminate with nul *)
 END MidStr;

(* Concat() - join string2 onto end of string1
 *)
PROCEDURE Concat(VAR string1, string2: ARRAY OF CHAR);
 CONST nul=0C;
 VAR i,j,k,l: LENSTR;
 BEGIN
    l:=HIGH(string1); k:=HIGH(string2);
    i:= Len(string1); (* locate end-of-string *)
    j:= 0;
    WHILE (i <= l) AND (j <= k) AND (string2[j] # nul) DO
      string1[i]:= string2[j]; INC(i); INC(j);
    END;
    IF (i <= l) THEN string1[i]:= nul; END; (* terminate with nul *)
 END Concat;

(* Compare() - compares string1 with string2; case sensitive.
 *            result:  -1 if string1 < string2
 *                      0 if string1 = string2
 *                     +1 if string1 > string2
 *)
PROCEDURE Compare(VAR string1, string2: ARRAY OF CHAR): INTEGER;
 CONST nul=0C;
 VAR i,j,l,k: LENSTR;
 BEGIN
    i:=0; j:=0; l:=HIGH(string1); k:=HIGH(string2);
    WHILE (i <= l) AND (j <= k) AND (string1[i] = string2[j]) DO
      INC(i); INC(j);
    END;
    IF ((string1[i] = nul) AND (string2[j] = nul)) OR
       ((i > l) AND (j > k)) THEN RETURN (0);  (* strings equal *)
 ELSIF (string1[i] = nul) OR (i > l) THEN RETURN (-1);  (* string1 < string2 *)
 ELSIF (string2[j] = nul) OR (j > k) THEN RETURN (+1);  (* string1 > string2 *)
    END;
 END Compare;

(* Asc() - ASCII code of the first char in string
 *)
PROCEDURE Asc(VAR string: ARRAY OF CHAR): CARDINAL;
BEGIN
   RETURN ORD(string[0]);
 END Asc;

(* String() - creates a string with all one char [c] of length [n]
 *)
PROCEDURE String(VAR string: ARRAY OF CHAR; n: LENSTR; c: CHAR);
 CONST nul=0C;
```

```
 VAR i,l: LENSTR;
 BEGIN
    i:=0; l:= HIGH(string);
    WHILE (i <= l) AND (i < n) DO string[i]:= c; INC(i); END;
    IF (i <= l) THEN string[i]:= nul; END;(* terminate with nul *)
 END String;

(* Spaces() - creates a string with all spaces of length [n]
 *)
PROCEDURE Spaces(VAR string: ARRAY OF CHAR; n: LENSTR);
 CONST sp=" ";
 BEGIN
    String(string, n, sp);
 END Spaces;

(* UCase() - modifies a string with ALL chars to uppercase
 *)
PROCEDURE UCase(VAR string: ARRAY OF CHAR);
 CONST nul=0C;
 VAR i,l: LENSTR;
 BEGIN
    i:=0; l:= HIGH(string);
    WHILE (i <= l) AND (string[i] # nul) DO
      string[i]:= CAP(string[i]); INC(i);
    END;
 END UCase;

(* LCase() - modifies a string with chars ["A".."Z"] to lowercase
 *)
PROCEDURE LCase(VAR string: ARRAY OF CHAR);
 CONST nul=0C;
 VAR i,l: LENSTR;
 BEGIN
    i:=0; l:=HIGH(string);
    WHILE (i <= l) AND (string[i] # nul) DO
      CASE string[i] OF "A".."Z" : INC(string[i],32); INC(i);
      ELSE INC(i);
      END;
    END;
 END LCase;

(* Trim() - trim trailing spaces from end of string; replace with nuls
 *)
PROCEDURE Trim(VAR string: ARRAY OF CHAR);
 CONST nul=0C; sp=" ";
 VAR i,l: LENSTR;
 BEGIN
    l:=HIGH(string);
    i:= Len(string); (* locate end-of-string *)
    WHILE (string[i] = sp) OR (string[i] = nul) DO string[i]:= nul; DEC(i); END;
 END Trim;

PROCEDURE Justify(VAR string: ARRAY OF CHAR; n: LENSTR); FORWARD;

(* Strip() - strip leading spaces from start of string; modifies string
 *)
PROCEDURE Strip(VAR string: ARRAY OF CHAR);
 CONST nul=0C; sp=" ";
 VAR i,l: LENSTR;
 BEGIN
   i:=0; l:=HIGH(string);
   WHILE (string[i] = sp) DO INC(i); END; (* skip over spaces *)
   Justify(string, i); (* left-justify starting from [i] *)
 END Strip;

PROCEDURE Delete(VAR string: ARRAY OF CHAR; n,len: LENSTR); FORWARD;

(* Justify() - left justify string starting from [n]th character
 *)
PROCEDURE Justify(VAR string: ARRAY OF CHAR; n: LENSTR);
 BEGIN
   Delete(string, 0, n);
 END Justify;

(* Delete() - deletes len characters from string starting at [n]th position
 *)
PROCEDURE Delete(VAR string: ARRAY OF CHAR; n,len: LENSTR);
 CONST nul=0C;
 VAR i,j,l: LENSTR;
 BEGIN
   i:=0; l:=HIGH(string);
   WHILE (i <= l) AND (string[i] # nul) AND (i < n) DO INC(i); END;
   j:= i + len;
   IF (j = i) THEN (* already left-justified *)
      WHILE (j <= l) AND (string[j] # nul) DO
        string[i]:= string[j]; INC(i); INC(j);
      END;
      IF (j <= l) THEN string[i]:= nul; END;  (* terminate with nul *)
   END;
 END Delete;

(* Remove() - remove a group of chars from start of string; modifies string
 *)
PROCEDURE Remove(VAR string: ARRAY OF CHAR; chars: ARRAY OF CHAR);
 CONST nul=0C;
 VAR i,j,l,k: LENSTR; match: BOOLEAN;
 BEGIN
   l:= HIGH(string); k:= Len(chars) - 1;
   i:= 0;
   REPEAT
     j:= 0; match:= FALSE;
     WHILE (j <= k) AND (chars[j] # nul) AND (match = FALSE) AND
           (i <= l) AND (string[i] # nul) DO
      IF (string[i] = chars[j]) THEN match:= TRUE; INC(i);
      ELSE INC(j); (* match:= FALSE *)
      END;
     END;
   UNTIL (match = FALSE);
   Justify(string, i);  (* left-justify starting from [i] *)
 END Remove;

(* NOTE ALSO:  DecStr(), HexStr() and BinStr() MUST contain enough space
 *             in "string", including preceeding sign char and trailing
 *             nul char !   (result "string" is left-justified).
 *)

(* DecStr() - a string representation in DECIMAL digits of INTEGER value [x]
 *)
PROCEDURE DecStr(VAR string: ARRAY OF CHAR; x: INTEGER);
 CONST nul=0C; minInt=32768;
 VAR i,j,l: LENSTR;
     xx: CARDINAL;
 BEGIN
    i:=0; l:=HIGH(string); j:= 1;
    string[i]:= nul; string[j]:= nul; (* terminate with nul *)
    IF (l >= 6) THEN (* else not enough space in array *)
      xx:= CAST(CARDINAL, x);
      IF (x < 0) THEN
        string[i]:="-"; xx:= minInt - xx;
      ELSE
        string[i]:=" "; xx:= xx - minInt;
      END;
      INC(i);
      REPEAT
        DEC(j); string[j]:= CHR(xx MOD 10 + ORD("0")); xx:= xx DIV 10;
      UNTIL (xx = 0) OR (j = 0);
      Justify(string, j); (* left-justify *)
    END;
 END DecStr;

(* HexStr() - a string representation in HEXADECIMAL of CARDINAL value [x]
 *)
PROCEDURE HexStr(VAR string: ARRAY OF CHAR; x: CARDINAL);
 CONST nul=0C;
 VAR i,j,l: LENSTR;
     n,xx: CARDINAL;
```

```
 BEGIN
    i:=0; l:=HIGH(string); j:= l;
    string[i]:= nul; string[j]:= nul; (* terminate with nul *)
    IF (l >= 5) THEN (* else not enough space in array *)
      xx:= x;
      string[i]:=" "; INC(i); (* start with a space *)
      REPEAT
        DEC(j); n:= xx MOD 16; IF (n > 9) THEN INC(n,7); END;
        string[j]:= CHR(n + ORD("0")); xx:=xx DIV 16;
      UNTIL (xx = 0) OR (j = 0);
      Justify(string, j);  (* left-justify *)
    END;
 END HexStr;

(* BinStr() - a string representation in BINARY digits of CARDINAL value [x]
 *)
PROCEDURE BinStr(VAR string: ARRAY OF CHAR; x: CARDINAL);
 CONST nul=0C;
 VAR i,j,l: LENSTR;
     xx: CARDINAL;
 BEGIN
    i:=0; l:=HIGH(string); j:= l;
    string[i]:= nul; string[j]:= nul; (* terminate with nul *)
    IF (l >= 17) THEN (* else not enough space in array *)
      xx:= x;
      string[i]:=" "; INC(i); (* start with a space *)
      REPEAT
        DEC(j);
        string[j]:= CHR(xx MOD 2 + ORD("0")); xx:= xx DIV 2;
      UNTIL (xx = 0) OR (j = 0);
      Justify(string, j);  (* left-justify *)
    END;
 END BinStr;

(* StrVal() - return numeric value of string representation in DECIMAL
 *         or HEXADECIMAL digits, depending on [base = 10 or 16]
 *)
PROCEDURE StrVal(VAR string: ARRAY OF CHAR; base: CARDINAL): INTEGER;
 CONST nul=0C; sp=" "; maxInt=32767; minInt=32768; limit=minInt/8;
 VAR f: BOOLEAN; i,l: LENSTR; x: CARDINAL;
 BEGIN
    i:=0; l:=HIGH(string); f:= TRUE; (* assume positive number *)
    CASE (string[i]) OF
     | nul : string:="0";
     | sp  : INC(i); f:= TRUE; (* positive number *)
     | "-" : INC(i); f:= FALSE; (* negative number *)
    ELSE (* do nothing *)
    END;
    x:= 0;
    WHILE (i <= l) AND (string[i] # nul) AND (x < limit) DO
      x:= x * base;
      CASE (string[i]) OF
       | "0".."9"          : INC(x, (ORD(string[i]) - ORD("0")));
       | "A".."F","a".."f" : INC(x, (ORD(string[i]) - ORD("A") + 10));
      ELSE ; (* do nothing *)
      END;
      INC(i);
    END;
    IF f = TRUE THEN  (* could be signed hexadecimal *)
      IF x > maxInt THEN
        x:= x - minInt; (* much too positive, make it negative! *)
      ELSE
        x:= x + minInt;
      END;
    ELSE (* f = FALSE - definitely a negative number *)
      IF x > minInt THEN
        x:= x; (* much too negative, make it positive! *)
      ELSE
        x:= minInt - x;
      END;
    END;
    RETURN CAST(INTEGER,x); (* a type transfer *)
 END StrVal;
```

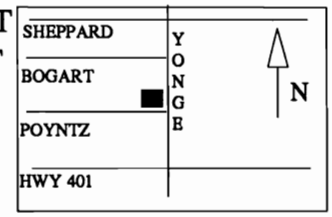
```
MODULE BaseConv;  (* Tiny Window for DECIMAL / HEXADECIMAL base conversions *)
                  (* written in M2Amiga by Anthony Bryant *)
FROM SYSTEM IMPORT ADR,ADDRESS;

FROM Conversions IMPORT StrToVal,ValToStr;

FROM Exec IMPORT WaitPort,GetMsg,ReplyMsg;

FROM Graphics IMPORT jam1,jam2;

FROM Intuition IMPORT
 Border,IntuiText,StringInfo,strGadget,GadgetFlags,GadgetFlagSet,
 Gadget,GadgetPtr,ActivationFlags,ActivationFlagSet,IntuiMessagePtr,
 IDCMPFlags,IDCMPFlagSet,WindowFlags,WindowFlagSet,NewWindow, WindowPtr,
 ScreenFlags,ScreenFlagSet,OpenWindow,WindowToFront,CloseWindow,RefreshGadgets;

CONST dec = 1; hex = 2;

VAR
 sg: ARRAY [dec..hex] OF Gadget;
 gb: ARRAY [dec..hex] OF Border;
 it: ARRAY [dec..hex] OF IntuiText;
 si: ARRAY [dec..hex] OF StringInfo;
 pts: ARRAY [1..10] OF INTEGER;
 buf: ARRAY [dec..hex],[0..12] OF CHAR;
 undo: ARRAY [0..12] OF CHAR;

PROCEDURE InitStrGadget(gID, top: CARDINAL; max: INTEGER; text: ADDRESS);
 BEGIN
  pts[1]:= 0; pts[2]:= 0;  pts[3]:=100; pts[4]:= 0;  pts[5]:=100; pts[6]:=10;
  pts[7]:= 0; pts[8]:=10;  pts[9]:= 0; pts[10]:=0;  (* common points *)
  WITH gb[gID] DO  (* could be common border *)
   leftEdge:=-2; topEdge:=-2; frontPen:= 1; backPen:= 0;
   drawMode:= jam1; count:= 5; xy:= ADR(pts);
   nextBorder:= NIL;
  END;
  WITH it[gID] DO
   frontPen:= 1; backPen:= 0; drawMode:= jam2;
   leftEdge:=-2; topEdge:=-10; iText:= text;
   iTextFont:= NIL; (* ROM fonts *)
   nextText:= NIL;
  END;
  WITH si[gID] DO
   buffer:= ADR(buf[gID]); undoBuffer:= ADR(undo);
   bufferPos:= 0; maxChars:= max;
  END;
  WITH sg[gID] DO
   nextGadget:= NIL;
   leftEdge:= 18; topEdge:= top; width:= 96; height:= 8;
   flags:= GadgetFlagSet{}; (* gadgHComp *)
   activation:= ActivationFlagSet{relVerify,stringRight};
   gadgetType:= strGadget; gadgetRender:= ADR(gb[gID]); (* gadget Border *)
   selectRender:= NIL; gadgetText:= ADR(it[gID]); (* intuiText *)
   specialInfo:= ADR(si[gID]); (* stringInfo *)
   gadgetID:= gID;
  END;
 END InitStrGadget;

VAR
 nw: NewWindow;
 nwp: WindowPtr;

PROCEDURE InitWindow(VAR nw: NewWindow; wtitle: ADDRESS);
 BEGIN
  WITH nw DO
   leftEdge:=320; topEdge:=70; width:=132; height:=56;
   detailPen:= 0; blockPen:=1;
   idcmpFlags:= IDCMPFlagSet{closeWindow,inactiveWindow,activeWindow,gadgetUp};
   flags:= WindowFlagSet{activate,windowClose,windowDrag,noCareRefresh};
   firstGadget:= NIL;
   checkMark:= NIL; title:= wtitle;
   type:= ScreenFlagSet{wbenchScreen};
  END;
```

```
 END InitWindow;

(* UCaseHex() - modifies a string with ALL valid HEX chars to uppercase
 *              and elliminates all non-valid chars, incl spaces.
 *)
PROCEDURE UCaseHex(VAR string: ARRAY OF CHAR); (* string is null-terminated *)
 VAR i,j: CARDINAL;
 BEGIN
   i:=0; j:=0;
   (* all valid HEX chars to uppercase, strip non-valid chars from string *)
   WHILE (string[i] # 0C) DO
     string[i]:= CAP(string[i]);
     CASE string[i] OF
       "0".."9","A".."F": string[j]:= string[i]; INC(j); (* left-justify *)
     ELSE END; (* do nothing *)
     INC(i);
   END;
   string[j]:= 0C;  (* terminate with null *)
 END UCaseHex;

PROCEDURE ProcessGadget(gID: CARDINAL);
 VAR v: LONGINT; s,e: BOOLEAN;
 BEGIN
  IF (gID = dec) THEN  (* v:= si[dec].longInt; *)
     s:= TRUE; (* decimal gadget is signed value *)
     IF (buf[dec,0] = 0C) THEN buf[dec]:="0"; END;
     (* NOTE: buf[dec] is already stripped of leading spaces *)
     StrToVal(buf[dec],v,s,10,e);
     ValToStr(v,FALSE,buf[hex],16,1,"0",e);
  ELSIF (gID = hex) THEN
     s:= FALSE; (* hex gadget is unsigned value *)
     UCaseHex(buf[hex]);
     IF (buf[hex,0] = 0C) THEN buf[hex]:="0"; END;
     StrToVal(buf[hex],v,s,16,e);
     ValToStr(v,TRUE,buf[dec],10,1,"0",e);
  END;
  RefreshGadgets(ADR(sg[dec]),nwp,NIL);
 END ProcessGadget;


VAR   (* main *)
 message: IntuiMessagePtr; class: IDCMPFlagSet; gadget: GadgetPtr;

BEGIN (* main *)
 (* init structures *)
 buf[dec]:="0"; buf[hex]:="0";
 InitWindow(nw,ADR(" BaseConv ")); nw.firstGadget:= ADR(sg[dec]);
 InitStrGadget(dec,21,12,ADR("Dec:")); sg[dec].nextGadget:= ADR(sg[hex]);
 InitStrGadget(hex,41,9,ADR("Hex:"));
 INCL(sg[dec].activation,longint); (* make integer string gadget *)
 nwp:= OpenWindow(nw);
 IF nwp # NIL THEN   (* process IDCMP messages *)
   REPEAT
    WaitPort(nwp^.userPort);
    LOOP
     message:= GetMsg(nwp^.userPort);
     IF message = NIL THEN EXIT END;
     class:= message^.class;
     gadget:= GadgetPtr(message^.iAddress);
     ReplyMsg(message);
     IF (closeWindow IN class) THEN
       EXIT;
     ELSIF (inactiveWindow IN class) OR (activeWindow IN class) THEN
       WindowToFront(nwp);  (* only executed once! *)
     ELSIF (gadgetUp IN class) THEN
       ProcessGadget(gadget^.gadgetID);
     END;
    END; (* LOOP *)
   UNTIL (closeWindow IN class);
   CloseWindow(nwp);
 END;
END BaseConv.mod
```


CANADA'S BEST SELECTION OF AMIGA - C64 - C128 SOFTWARE
COMSPEC HARD DRIVES AND BERNOULLI DRIVES
AVAILABLE FOR
THE AMIGA 500 AND 2000

Electronics 2001
COMPUTER PRODUCT CENTRE

4736 YONGE STREET
WILLOWDALE ,ONT
M2N 5M6
TEL: (416) 229-2700



AUTHORIZED COMMODORE DEALER SINCE 1979

D - Five Associates
19 Crosby Drive
Bedford, MA
01730-0523
(617) 275-8892

C-64/128 & peripherals

Tired of the high cost of computer repairs?
→ FLAT Labor charges
→ FREE Estimates
→ Warranty work

Commodore PC-10

Amiga 1000

* plus parts and sales tax

Also:
1764 to 512K: $61.95
128 64K vdc RAM: $40.00
NEW: C=1902 conversion to RGB-I: $40.00

Authorized Commodore Service Center

MCP Associates Inc.
Featuring C.Itoh, Sony, Etc:
The DISK SPECIALISTS— DISTRIBUTORS
3.5", 100% Certified, Lifetime Warranty!
SALE - Limited time offer

Bulk—C.Itoh Prices per disk Based on 100 disks	SSDD—Blue	DSDD—Blue	DSDD—Rainbow
	0.95	1.19	1.29

Sony, other brands & products available. Call/Write for Disk Pricing Schedule & Amiga Product Listings. Prices subject to change without notice. S/H—$5.50; COD's add $3.00. Call for details. Educational, corporate & dealer inquiries invited.

MCP Associates, Inc.,
PO Box 6260, Dept AW, L.I.C., NY 11106-0260
Tel: 1-718-956-9000 Fax: 1-718-956-9028
Exceptional Service & Quality

New Store!
Fisher's
Computers & Software
Authorized Amiga Dealer
8005 Archibald
Rancho Cucamonga, CA 91730
(714) 987-1662
Serving The Inland Impire

Reserve

Create a memory shortage on your Amiga!

by Nick Sullivan

One essential phase of testing a new Amiga program involves making sure that it can handle the low-memory conditions that can easily arise in a multitasking system. The *Reserve* utility listed below was written to make it easier to perform low-memory testing in a controlled way. With its aid you can quickly test a program under memory shortages of various levels of severity, then - if your program survives! - return to your full-memory environment for editing and recompiling/reassembling as required.

```
/* reserve.c

   by Nick Sullivan
   Kwik Filler Enterprises, February 20, 1989

   This program is designed for use in development as an aid in verifying
   that your program gracefully handles memory problems. Run your program
   first, then run Reserve, telling it how much CHIP memory (in 1000s of
   bytes) you would like it to leave unallocated, e.g.:

      Reserve 100

   would allocate all free memory on the machine except for 100,000 bytes of
   CHIP RAM. When you are finished testing, click the close gadget on
   Reserve's window; it will exit after freeing all the memory it has
   allocated. You can also use Reserve to allocate all FAST memory, leaving
   CHIP memory unaffected (except as needed by Reserve itself), with:

      Reserve 0

   This source code is in the public domain.
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <ctype.h>

struct IntuitionBase *IntuitionBase;
void *AllocMem(), *OpenLibrary(), *GetMsg();
struct Window *OpenWindow();
long atol(), Output();

struct NewWindow nw = {
   0, 11, 150, 10, 0, 1, CLOSEWINDOW,
   WINDOWCLOSE | WINDOWDRAG | WINDOWDEPTH, NULL, NULL,
   (UBYTE *)"Reserve", NULL, NULL, 0, 0, 0, 0, WBENCHSCREEN };

struct Window *win;
long  memreqs;

main (argc, argv) int argc; char **argv;
{
register long rsize, size, *p, *oldp;
struct IntuiMessage *Imsg;
char *rsv;

   /* check that there is a command line arg, and it starts with digit */
   if (argc != 2 || !isdigit(*argv[1]))
      CloseStuff("Usage: reserve SIZE (CHIP memory to reserve, in K)\n");
   /* if arg is zero, this is 'nofastmem' mode */
   if (!(rsize = atol(argv[1]) * 1000L)) memreqs = MEMF_FAST;

   /* open Intuition and a little window, and allocate reserved mem */
   if (!(IntuitionBase = OpenLibrary("intuition.library", 33L)) ||
         !(win = OpenWindow(&nw)) ||
         (rsize && !(rsv = AllocMem(rsize, MEMF_CHIP))))
      CloseStuff("Out of memory\n");

   /* allocate all memory in the system, and write the address of the
      previous block allocated at the beginning of the current one. */
   size = 0x100000;        /* start by allocating very large blocks */
   oldp = NULL;            /* no blocks previously allocated yet    */

   while (size >= 8) {     /* keep going till even small allocations fail */
      while (p = AllocMem(size, memreqs)) {
         *p       = (long)oldp;     /* save address of previous allocation */
         *(p + 1) = size;           /* and size of this one for later use  */
         oldp = p;
      }
      size >>= 1; /* can't allocate more at current size; halve & continue */
   }

   /* allocations all done - now free the reserved block */
   if (rsize) FreeMem(rsv, rsize);

   /* wait for a message at the window - only CLOSEWINDOW possible */
   WaitPort(win->UserPort);
   while (Imsg = GetMsg(win->UserPort))
      ReplyMsg(Imsg);

   /* now free all the allocated memory and shut down */
   p = oldp;
   do {
      p = (long *)(*oldp);
      FreeMem(oldp, *(oldp + 1));
   } while (oldp = p);
   CloseStuff(NULL);
}

CloseStuff (txt) char *txt; /* shut down everything and free resources */
{
   if (txt)           Write(Output(), txt, (long)strlen(txt));
   if (win)           CloseWindow(win);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   exit(0);
}
```

The BEST AMIGA DISKS

Quality User Supported Software works with all Amiga systems

$7 Each Buy 1-4 Disks

$6 Each Buy 5-14 Disks

$5 Each Buy 15 or more Disks

So easy to Use

- ▼ No Computer Experience Necessary
- ▼ Instructions for each Order
- ▼ FREE Phone Support

So Easy to Order

- ▼ FREE Membership ▼ FREE 800-# for Orders
- ▼ FREE Same-day Shipping
- ▼ UPS 2nd Day Air Service when you need it Now!
- ▼ FREE Catalog ▼ Your Satisfaction Guaranteed

THE TOP 10

☐ **#131 PacMan '87** - Great sound and graphics. Adds new elements to PacMan. Saves Top 10.
☐ **#127 Wheel of Fortune** - A great computer version for multiple players. It even talks.
☐ **#23 Monopoly** - Enjoy great graphics and sound while playing three tough computer opponents.
☐ **#37 Business Programs** - Included are an address book, an amortization program, a talking mail manager, and a label printer.
☐ **#27 Amoeba Invaders** - A better Space Invaders!
☐ **#140 Virus Killer** - Everyone needs this! Makes it easy to detect and eliminate known viruses.
☐ **#115 Word Processor** - Lots of features.
☐ **#134 Applications** - Label maker/printer, grocery list maker, and AMIGazer - a star viewing program.
☐ **#142 Q-Bert** - Like the popular arcade version.
☐ **#139 Bull Run** - Great Civil War strategy board game, with impressive graphics and sound.

1-800-444-5457 for Orders

THE BEST OF THE REST

503-664-5953 - Foreign Orders

BUSINESS/HOME

☐ **#116 Business II** - VC-Spreadsheet, HP-10c calculator, and several diversions for when the boss isn't looking!
☐ **#117 Business III** - DataBase, a bunch of great new fonts, RSLClock-great clock utility, AmigaSpell.

LANGUAGES

☐ **#9 FORTH** - Two versions of the FORTH programming language.
☐ **#50 XLISP** - A version of LISP, the artificial intelligence language. Includes documentation.

COMMUNICATIONS

☐ **#4 Communications** - Starterm and Aterm are both included. Both run from Workbench or CLI and are comparable to communications packages selling for $50+. Full control of baud rates, phone directory, all protocols, auto chop of files, and many other extras. Works well with the Avatex 1200 modem and others. Source code included.
☐ **#90 Modem Madness!** - Terminals include StarTerm, ATerm, and Kermit. Telecommunication utilities such as Archive are also here. Other types of programs and utilities are also included in the price of admission.

UTILITIES/APPLICATIONS

☐ **#60 AmigaBasic** - Two programs that are truly of commercial quality, Cell-Animate and Graphit. Some Deluxe Paint picture files are also included.
☐ **#97 Tutorial Disk1** - A disk full of information and programs to instruct Amiga Programmers and users. Several C & ASM source files are included.
☐ **#98 Tutorial Disk2** - More of the best of Amiga Information.
☐ **#101 Utilities** - Many new utilities like Timeset - a time setting utility and DirCopy-a great copier (very quick), and about a dozen more. Some new fonts are also included on this disk.
☐ **#105 Potpourri I** - This disk contains several different kinds of programs, some of the highlights are: PopCLI2-evokes a new CLI window at the press of a button; PSound-sample sound recorder and editor; 3-D Breakout; DiskCat-catalogs and organizes disk files; IconMaker-makes icons for most programs so that they can run from Workbench; FKey-template maker.
☐ **#129 Amiga Utilities II** - A hard disk backup; Target-sounds a gunshot whenever the left mouse button is pressed; Dpaint Tutor; WinSize-change window size from CLI easily, and lots more.
☐ **#132 Videomaker Utilities** - This disk is packed with utilities to make your desktop videos easier to produce and more professional looking.
☐ **#133 DOS Helper** - A program designed to help you with the AmigaDOS commands. Can be activated from icon of the CLI. Supports multitasking, so that you can refer to it when you need it. As usual, there are other good programs included on the disk.
☐ **#135 Applications II** - Long Movie-plays several IFF pictures in fast succession, creating animation. QuickBase-a mail manager DBase. Persmait-a DataBase for keeping records of friends, family, associates, customers or employees. MORE.
☐ **#146 Calendar** - A very good personal calender for birthdays, holidays, meetings, bills and other events. Excellent graphics. Calendar program also has a diary. Other programs include some graphics and Checkbook.
☐ **#150 Textcraft+ Demo** - A demo of a very good word-processing program of high quality. Menu driven. Has a lot of helps to show you how. Textcraft+ also has a Speller Check available.

SOUND/MUSIC

☐ **#18 Future Sound Demo** - Another great sound demo of digitized sound. Includes the wicked witch of OZ, breaking dishes, sea gulls, car crash, ducks, others.
☐ **#30 Super Sounds** - Great digitized sounds from movies like Star Trek, 2001, James Bond movies, Star Wars, and Starman, Is it real or is it the Amiga?!!!
☐ **#77 Instruments** - Turn your keyboard into 25 different musical instruments.

SLIDE SHOWS

☐ **#1 Norman Rockwell** - 17 beautiful digitized Rockwell paintings in this self-running slideshow presentation. You've got to see these!
☐ **#55 EA Demo and Polyscope** - More great graphics for your enjoyment.
☐ **#67 DPSlide8** - Over 30 slides of all sorts on this self-running slideshow.
☐ **#94 Diga-View** - This one shows the digitizing process in stages. Several good pictures are included. Other programs also included.
☐ **#95 DigaSlide11** - Another in the great series of slideshows with great artwork. Self-running with over 25 pictures.
☐ **#108B Juggler** - Famous demo that shows the beautiful graphics of the Amiga and just how powerful this program is. It is easier to run than 108A, but only has the one demo on it.

GAMES

☐ **#38 Basic GrabBag2** - Around 25 programs of various types. Many of these are must-haves. At less than $.20 each, you can't go wrong!
☐ **#44 Games3** - More great games including Life, Vegas Slot Machine, Reversi, others.
☐ **#52 Basic Games** - Tons of ABasic games - discover some treasures!
☐ **#61 ABasic GrabBag1** - Only about 100 of all types!!!!!
☐ **#102 Sinking Island** - Return to Sinking Island is an excellent adventure game. Well worth the price - hours of enjoyment!!!
☐ **#114 Potpourri X** - Othello, A key-shortcut program for AmigaDos. Various new tools, automatic printer-driver generator, much more.
☐ **#118 Great Graphic Games** - Includes Missle Command, 2-D Triclops, Cosmo-asteroids clone, BrakeOut, Yatzee, Hack and more.
☐ **#121 Backgammon** - A great game from David Addison.
☐ **#122 Solitaire** - Two versions by David Addison.
☐ **#123 Cribbage** - Take on the computer or a friend.
☐ **#124 Milestone** - A great computer version of Miles Bournes by the author of Monopoly for the Amiga, David Addison.
☐ **#125 Othello** - A great 3-D version of this popular game.
☐ **#128 Space Games** - Cosmoroids (like asteroids) and Gravity Wars highlight this disk just full of games.
☐ **#137 Blackjack** - A full-featured game which allows pair-splitting, double-down, etc. Bandit-play the slot machines without going to Vegas!!! More.
☐ **#141 Dominoes** - Dominoes game with great graphics. Also Tic Tac Toe, Drawing and Molecules programs.
☐ **#147 Jackland/Graphics** - Adventure clue game. Also some great pictures (graphics), a useful utility called Quickbase, and a fun program called Thingies which you will enjoy!!!
☐ **#148 Boulder Dash** - Very popular game with excellent graphics and has several challenging levels. This disk is full- It has Othello, Life3 and many useful utilities.
☐ **#151 4 in a Row** - A fun, but challenging game you play against the computer. There is an excellent Demo "MandFXP-D3", a utility or 2 and the fun TARGET - A weirdo thingy.

MISCELLANEOUS

☐ **#88 Amiga Basic Programs** - Over 50. Games, utilities, applications, entertainment, and finance. Also included is a program that allows you to use IFF files in your Amiga Basic programs.
☐ **#119 mCAD** - A full-featured computer-aided design program.
☐ **#136 Graphics2** - Border Set-useful for desktop publishing and video, making cards, coupons or menus, and your own artwork. Xicon-allows you to run AmigaDOS commands or programs from Icon.
☐ **#144 Christmas Animations** - 10 beautiful scenes and graphics with sound. Great scenes include: Lit Candle, Elves, The Christmas Tree, The Manger, Season's Greetings, Holly Wreath, Chimney Smoke, Church Bells, and Walking Home.
☐ **#145 Animations 3** - More Great Animations. 3 very good demos plus 3 workbench pictures and Blobs.

Name _____
Address _____
City _____ State ____ Zip ____
Phone (____) _____
Visa/MC ____-____-____-____
Sig. _____
Exp. Date _____ ☐ Send Catalog

Disks Ordered ____ x ____ $____
☐ SHIPPING Free U.S. $____
Canada - 25¢ per disk Foreign - 50¢ per disk
☐ COD $____
(add $4 if you require COD)
☐ U.P.S. 2nd Day Air (Add $3) $____
TOTAL $____
☐ Check/MO ☐ Visa/MC ☐ COD

Software Excitement!

P.O. Box 5069 • Central Point, OR 97502

VISA ORDER TODAY MasterCard

1-800-444-5457

Changes In The Disk Filing System

Another look at file structure - this time for FFS

by Betty Clay

We are all aware of the new Fast Filing System (FFS). Comparisons of the speed of the old and new systems turn up everywhere. Except for floppy disks, the new system is an unqualified winner. For the average user of the filing system, this is enough information. It works. It's faster. "Off with the old; on with the new." Some people even insist on using the new system on floppies, for which it is not supported, is not always reliable, and isn't even faster. I have great curiosity about filing systems, and want to know what is there. When there are changes, I want to know exactly what those changes are and how they affect my data. If I choose to read a block with a disk editor, I want to know which data is system information and which is part of my file. A new filing system meant that there had been changes to the file system information covered in my previous articles (*Transactor,* Volume 7, Issues 3 and 6). This one deals mostly with those changes, with enough general information to make it readable to those who have not spent a great deal of time nosing around in the various disk sectors. Having no hard disk on which to use it, and unwilling to use it on floppies for which it is not recommended, I set up a recoverable RAM disk with 80 cylinders, formatted it with FFS, and used it for this study.

What kind of disk am I?

The first thing the system reads on a disk or on a hard disk partition is the boot block. This block will contain a long word that identifies the disk as a DOS disk, and, on a floppy, this may be followed by the program needed to boot the system. There is considerable room here - two sectors of 512 bytes each, less the 32 bits that identify the DOS type. If a disk is freshly formatted and then installed, only a few of these bytes are used. The remainder has been used for protection by many software developers, and for storing viruses by people with even less constructive minds. Hard disk owners have been free of concerns about boot block viruses for the most part, because the hard disk driver takes care of booting, and only the 32-bit identifier was needed. If this identification number is corrupt, the system will notify you that this is not a valid DOS disk, and restoring this number is a good place to start recovery efforts.

In the boot blocks, the only real difference noted between the old file system and the new is in the last byte in the opening long word - $444F5300 for the old system, and $444F5301 for the fast system. These are the ASCII characters 'DOS' plus a 0 or 1 respectively. When the system reads this number, it knows which filing system to use; if it finds neither of these in that position, you will get the "Not a DOS disk" requester. In fact, it has been possible to rescue some entire hard disk partitions by merely restoring the value of that one long word.

Here is what I found in the two bootblocks when I examined them with DiskED. (I greatly prefer DiskEd for this kind of study, since it clearly marks things out by position. I do wish Commodore had polished it a bit and released it to a more general audience.)

```
# g 0        ; get the first bootblock
Block 0 read (cyl 0, sur 0, sec 0)
# sx         ; set to read in hex
00000000
# t 0 10       ; type the first ten longwords
0: 444F5300
1: C0200F19
2: 00000370
3: 43FA0018
4: 4EAEFFA0
5: 4A80670A
6: 20402068
7: 00167000
8: 4E7570FF
9: 60FA646F
10: 732E6C69
```

and for the FFS:

```
# sx           ;set to display hex
00000260
# g 0          ;get the first bootblock
Block 0 read (cyl 0, sur 0, sec 0)
# t 0 10         ; type the first ten words
0: 444F5301
```

```
1: 444F5302    ; different because RAD: had
2: 444F5303    ; not been 'installed'.
3: 444F5304
4: 444F5305
5: 444F5306
6: 444F5307
7: 444F5308
8: 444F5309
9: 444F530A
10: 444F530B
```

The device being examined is the RAM-based 'RAD:'. Attempts to install RAD: were unsuccessful, so the boot block of RAD: does not hold the booting sequence in these examples. The key, though, is in that first number of each sequence.

At the root of it all

The root directory is the same for both old and new systems, or so it would seem at first glance. Not entirely so! The root block is the key to the entire disk, in either kind of system. It is the centermost sector in any kind of disk or partition, thus making seek time as small as possible for every sector. Other blocks on the disk are accessed through this block, and if it becomes corrupt one must take drastic measures to recover it or else give up the disk. Upon boot-up, the system checks the boot block, then the root directory. The boot sequence on a floppy tells the file system where to find the root, so the driver of a hard disk will have to do this also.

In the root directory on either system, the first six slots (32 bits each) have system information - the kind of block (short, a single sector), two unused slots, the size of the hash table (72 slots for a 512 byte sector), a slot reserved for future use, and a checksum. This is followed by a table of seventy-two slots (32 bits each) that hold the numbers of sectors in which you will find user-directory blocks or file header blocks. The system software is written so that it would still work if the size of the blocks were changed, but no such change has been announced. There is a hint that changes might occur in 1.4, however, so programmers should be aware that we might not always have this standard 512 byte size, and should write software that can handle other sizes. Most software does not need to be concerned with the size, but those who write disk recovery programs or disk editors will need to make allowance for these differences that may (probably will) occur in future systems.

While the hash table is not a major emphasis in this article, it is worth noting that the hash values must currently fall between 6 and 77, inclusive. The lowest value is six because of the system information in slots zero through five. The hashing algorithm has appeared in messages written by Commodore employees on several occasions:

```
Hash(name)
unsigned char *name
{
int val;
  val = strlen(name);
  while (*name)
    val = ((val*13) + (int)toupper(*name++)) & 0x7ff
  return (val % 72);
}
```

Every possible file name will come out of this algorithm with a value within this range, and its header block will be placed in the slot corresponding to its hash value. In the case of two files with the same hash value, provision is made through a hash chain.

After the hash table, there is one longword that holds the bitmap flags, and which should contain either -1 or 0. When the bitmap has finished its update, this flag is set to -1; should the system come up and find a 0 there, the validator will go to work to rebuild the bitmap. This can take considerable time, and we do notice when it happens. I have read of people with hard disks having to wait minutes for the validator to finish, and even floppies can take lots of time and thrashing about. Most of us have learned to wait for that last little write now, but we still get impatient occasionally, and pay dearly for our haste the next time we use the disk.

In the next two fields, we find a difference between the old and new systems again. In the old system, the BitMapKeys and BitMapExtension fields are reversed, and the BitMapKeys are used from position #24 up. In the FFS, immediately following the bitmap flags, there is a set of 25 positions in which to store the numbers of the sectors that hold the actual bitmaps. Each position can store the information for 32 sectors, so on floppies, one sector was always adequate, and the rest of these slots were empty. With hard disks, it takes more, and these are provided for with the BitMap extension. Should the last position in the bitmap table contain something other than zero, the system knows there will be more bitmap blocks listed, and it consults the BitMapExtend block to find them. This is a 32-bit slot containing the number of the block that holds the overflow. A BitMapExtension block does not have even so much as a checksum to take up space. It consists of 127 block numbers and the number of the next extension block. Since all of these lists are read from the bottom to the top, the first number in this sector would give the number of the next extension block.

Here are some excerpts from readings of OFS and FFS disks. The disks are not identical, so some differences in numbers are inherent.

```
# sd
2
# t 75 90
75: 0
76: 1357  ; From OFS Workbench:
77: 0
78: -1    ;bitmap is valid
79: 1029  ;Bitmap is in sector 1029
80: 0
```

```
81: 0
82: 0
83: 0

# g 880
Block 880 read (cyl 40, sur 0, sec 0)
# t 75 85
75: 0   ;From FFS Workbench in RAD:
76: 0
77: 0
78: -1   ;Bitmap is valid
79: 881  ;Bitmap is in sector 881
80: 0
81: 0
82: 0
83: 0
```

A correction to this part of the root block permitted Commodore to remove the 53 megabyte limit on partition size which plagued the old file system. The old system assumed that the root had protection bits, as did the other directory blocks, and it proceeded to write those bits into the field that pointed to the bitmap extension block. This happened because the protection bits in user directories occupy the position used for the extension block number in the root directory. In the FFS, this bug has been corrected, but users of the old file system will continue to have this 53M limitation.

The remaining portion of the root directory remains the same in both systems: the date of the last alteration of the root block, the disk name, the date of the last alteration of a file or partition (though this is still not working quite right), the date the disk was made, some slots reserved for future use, and a longword that identifies the kind of block (secondary type). In the Fast File System, there is still an error in 1.3 that causes the date of the last alteration of the file or partition to repeat the date the root directory was last altered, rather than giving the date of alteration of the partition or file.

The actual bitmap blocks, unlike the extension blocks, do have a checksum. Remember that the extension blocks hold only other sector numbers. The bitmap blocks indicate which sectors are used and which are free. A bitmap block will have the first slot occupied by a checksum, followed by 127 longs, each representing 32 sectors, or 4064 sectors per block. This is not quite two megabytes, in case you want to compute the number of bitmap blocks needed for your disk or partition. If a bit is set to 1, the block to which it corresponds is available for use. When used, the bit is changed to zero. At first, this seems to be backwards. Upon reflection, we can see that to have used zero to indicate a free block would have required a search of every bit individually, but testing for non-zero allows the testing of 32 bits at a time. If a non-zero condition is found, then that long may be searched to see which block is available. This is faster than checking every long for a zero bit.

To the best of my knowledge, no change has been made to the user directory blocks in the fast file system. The first 78 slots appear to be the same as in a root directory. The user directories have no need of bitmap information, so it is replaced with the unused positions (seven of them, but not contiguous), the protection bits, and a comment field of which 80 characters are available to the user. There is only one date field in a user directory - that of the date it was created. The user directory name field has space reserved for 36 characters, rather than 40, but this is more than enough, since the name in both cases is limited to 30. A user directory can be on the hash chain, so there is a position to hold the number of the next block on that chain, one to hold the number of the parent directory's header block, and a secondary file type number.

The hash chain is interesting. Before coming to the Amiga, I used the much simpler Commodore computers that had a single directory track (or, in the case of the 8050, two tracks). These were located, as is our root, in the center of the disk, and each contained the directory information about all files on the disk. It was very easy to read the directories quickly, but it was also easy to run out of space to add more files. Each directory track could hold a finite number of files, and once it was filled, the disk was full no matter how much storage space might still be available. In the Amiga, the hash chain has removed this problem. If two file names get the same value when put through the hashing routine, the first one is placed in the root directory, and the next is referenced through the hash chain position in the header of the next file on the chain. Thus, if my first file were placed in slot nine of the root, when I saved a different file with a hash value of nine, the system would go to position nine in the root, read the sector referenced there, and compare the name of that file with the one for which I asked. If they are different, the slot for the hash chain would be checked, and the sector referenced in the hash chain slot would be read. The checking process would continue until my file was found. The last file on the chain has a zero in this position, so the file system knows when to quit.

So what's really new?

The changes in the boot, directory, and bitmap blocks are small. It is not until we come to the data blocks that the changes in the FFS really become apparent. A quick glance at any of the other blocks would not show any real difference at all - though some have been made, and have been noted above. Even the file header block shows no large difference. According to a Usenet message, the number of blocks in the file (third position in the header block) is not updated by the fast file system, but it is still there.

The next item in the file chain is the file header blocks. They serve several purposes. For one thing, they are used by the FileLock structures. Commodore has told us that the FileLock system will continue to be supported in the future. When a file is locked, the FileLock field 'fl_Key' contains the number of the header block on which the file begins on a disk or partition, or, in the case of a RAM disk, the address at which the file is currently stored. The header block contains the necessary information about a file: its name, when it was created, its

protection bits, and pointers to the blocks that hold the data and to extension blocks.

File header blocks use the first six positions for such system information as type of block, number of this block, number of blocks in the file, the number of the first block that holds data, and a checksum. This is followed by yet another table of 72 positions, each of which holds the number of one data block used by this file. The slots are filled from #78 up to #6. Should the file require more than seventy-two sectors, the number of the extension block will be placed in slot 126. After the table, there will be the usual spare slots reserved for future use (three of them in this case), and those needed for the date and file name, a hash chain entry, the number of the parent directory, the extension block number, and the secondary file type (-3 in this case).

When we arrive at the actual data blocks, however, we find one really big change. The first six longwords in the old system contained system information much like that in the directories. This left room for only 488 bytes of data, since the other 24 were used for the system. Not only did this take up extra room on the disk - it slowed down the file system.

Using Disked, let's find a file under each system, and then compare the two files. The hash value of 'c' is 14; that of 'run' is 6. So let's start at the beginning, find the 'c' directory, and trace it down to the 'run' command under each system:

To locate the 'c' directory, we first read the root block:

```
Fast File system:        Old File System:
0: 2                 0: 2     ;Type of block (short)
1: 0                 1: 0
2: 0                 2: 0
3: 72                3: 72    ;size of hash table
4: 0                 4: 0
5: -1199734386       5: 9069645323  ;checksum
6: 0                 6: 0
7: 0                 7: 882
8: 0                 8: 0
9: 0                 9: 0
10: 0                10: 884
11:                  11: 0
12: 0                12: 0
13: 0                13: 885
14: 882              14: 887  <--block containing C
directory
```

Next, we read the sector that holds the C directory, and look for the item in slot 6, which is the hash value of 'run'.

```
FFS#                 OFS
# g 882        # g 887    ;Get the block where C is
Block 882 read              Block 887 read
(cyl 40, sur 0, sec 2)      (cyl 40, sur 0, sec 7)
# t 0 10       # t 0 10    ;type the 1st ten positions
0: 2               0: 2    ;kind of block (short)
1: 882             1: 887  ;number of this block
2: 0               2: 0
3: 0               3: 0
4: 0               4: 0
5: -23315566       5: -23327944 ;checksum
6: 883             6: 888  <-header block for 'run'
7: 885             7: 890
8: 889             8: 897
9: 891             9: 899
10: 895            10: 901
```

The header blocks are the same for both systems. It is necessary to visit this block, however, in order to find the numbers of the sectors that hold the actual data for 'run'. Next, we move to block 883 on the FFS; block 888 on the old system, which are the file header blocks for 'run':

```
FFS                  OFS
0: 2                 0: 2     Kind of block (short
1: 883               1: 888   Number of this block
2: 6                 2: 6     Sectors used in this file
3: 0                 3: 0     Data blocks for this file
4: 884               4: 889   First block of the file
5: -55752819         5: -55752929
6: 0                 6: 0
7: 0                 7: 0
...........          ...........
71: 0                71: 0
72: 906              72: 873  Last block used by 'run'
73: 905              73: 872
74: 904              74: 871
75: 903              75: 870
76: 902              76: 879  Second block used by 'run'
77: 884              77: 889  First block used by 'run'
78: 0                78: 0
```

And when we read block 884 of the FFS, or block 887 of the OFS, we finally see the major difference between the two systems. Here I changed from decimal to hexadecimal notation, to show more clearly exactly what is there.

```
FFS:                 OFS:
# t 0 20
0: 00000008                 Kind of block (data)
1: 00000378                 Number of this sector
2: 00000001                 How far into the file?
3: 000001E8                 Amt. of data (488 bytes)
4: 0000036F                 Next block in file (879)
5: 7CDCFCB7                 Checksum
0: 000003F3   6: 000003F3   <hunk_header, the begin-
1: 00000000   7: 00000000    ning of the load info
2: 00000002   8: 00000002
3: 00000000   9: 00000000
4: 00000001  10: 00000001
5: 0000004F  11: 0000004F
6: 00000226  12: 00000226
7: 000003E9  13: 000003E9    <hunk_code
8: 0000004F  14: 0000004F
```

```
 9: 286A0164  15: 286A0164
10: 700C4E95  16: 700C4E95
```

As you can see, the data is identical, but the old file system has 24 bytes of information that is not used in the fast system. This speeds things up in several ways. Obviously, there is more data on each sector, requiring fewer reads. Less obviously, there are fewer steps needed to make the data ready for use. Under the OFS, the track is read into the track buffer and decoded, then the 24 bytes at the beginning of each sector are stripped off before the information is copied into the user's buffer. With the new system, that entire step can be omitted and the data is DMA'ed directly into that buffer. To have this saving for every sector can make a very large difference in the amount of time required. Even more significantly, hard drive controllers can transfer data very quickly, but there is a certain amount of overhead to request a block of data. It is much faster to request, say, four blocks at once than to ask for each block in turn.

In order to gain the saving in time and space, we lost some means of recovering lost files. At first glance, it might seem that there is no way at all to recover, but this is not totally true. In addition to the file blocks that we see, there are label areas for each of these sectors. I have not found documentation for the label area under FFS, but under the old system, the RKM told us that this area contained:

2 bytes of 00
2 bytes of A1 (making a sync byte for MFM)
1 byte to indicate the format type
1 byte for the track number
1 byte for the sector number
16 bytes that were not used under the old system (and I believe not used in the new system either), but intended for future use as file recovery information
4 bytes for the label checksum
4 bytes for the data block checksum

The track, the sector, and the checksum, then, are still here in the sector label that we don't see in our disk editors. It was redundant to have this repeated in the first six words of the sector itself, though it did prove useful for disk recovery and file integrity. The data blocks no longer have the pointers to their parent file, and we are now dependent on the list blocks if we must recover data.

And down the road?

Future releases of the operating system will see far greater changes than those in 1.3. A study of the layout of the system told us from the beginning that we would not always have 512 byte data blocks, and it is quite likely that variable sized blocks will be recognized as early as the 1.4 release. Larger file blocks will call for fewer reads and fewer seeks, making the system faster. Hash tables will eventually be longer than 72 slots, causing fewer hash chain collisions. The basic system for both of these is already laid out, and can be implemented with minimum disruption. Other changes will follow. Software that accesses the disk in the approved manner will continue to work. That which bypasses the correct methods to gain temporary advantage will break when the system is changed.

C structures for the various disk blocks (based on those published in AmigaMail, November/December 1988):

```
struct RootBlock{
 LONG  Type;            /*(set to short (2)*/
 ULONG OwnKey;          /*unused in root, so set to 0*/
 ULONG SeqNum;          /*unused*/
 ULONG HtSize;          /*number of slots in hash table*/
 ULONG Nothing1;        /*reserved for future use, so 0*/
 LONG  Checksum;
 ULONG HashTable[72];   /*the actual hash table with 72 slots*/
 LONG  BitmapFlag;      /*-1 if valid; 0 if not validated*/
 ULONG BitmapKeys[25];  /*for FFS - OFS has BitmpaExtend here*/
 ULONG BitmapExtend;    /*where is the next list of BitmapKeys?*/

     For FFS only. OFS had a problem with
     this slot, as explained in the text.

 ULONG DirAltered[3];   /*datestamp for last alteration*/
 char Name[40];         /*BCPL string, name of disk*/
 ULONG DiskAltered[3];  /*currently has a bug in FFS, intended*/
                        /*to show last date any file on this disk*/
                        /*was altered*/
 ULONG DiskMade[3];     /*Date when disk or partition was*/
                        /*formatted*/
 ULONG Nothing2;        /*"Nothing" slots are reserved for*/
 ULONG Nothing3;        /*future use*/
 ULONG Nothing4;
 LONG SecondaryType;    /*Type 1, secondary file type is ROOT*/
}


struct UserDirectoryBlock {
 LONG  Type;      /*short (2)*/
 ULONG OwnKey;    /*number of this sector*/
 ULONG Spare1;    /*all spares are set to 0, and are
                  /*reserved for future use*/
 ULONG Spare2;
 ULONG Spare3;
 LONG  Checksum;
 ULONG HashTable[72]; /*number of slots available in this table*/
 LONG  Spare4;
 LONG  Spare5;
 ULONG Protection;    /*Protection bits for the directory*/
 LONG  Spare6;
 char  Comment[92];   /*but only 80 available for the user*/
 ULONG Created[3];    /*Date the directory was created OR*/
                      /*modified*/
 char  DirName[36];   /*directory name, but only 30 are used*/
 LONG  Spare7[7];
 ULONG HashChain;     /*where is next file with this same*/
                      /*hash value? (0 if no more exist)*/
 ULONG Parent;        /*Block where the parent directory is*/
 ULONG Spare8;
 LONG SecondaryType; /*Type 2, ST_USERDIR*/
}


struct FileHeaderBlock {
 LONG  Type;            /*short file, so 2*/
 ULONG OwnKey;          /*the number of this same block*/
 ULONG HighSeq;         /*how many blocks used for this file?*/
                        /*used for old system, not updated by FFS*/
```

```
 ULONG DataSize;        /*number of data blocks for this file*/
 ULONG FirstBlock;      /*what sector does this file start on?*/
 LONG Checksum;
 ULONG DataBlocks[72];  /*list of up to 72 sectors used by*/
                        /*this file, beginning with #71, moving*/
                        /*back up to #0, then moving to the*/
                        /*extension block if more were used*/
 ULONG Spare1;
 ULONG Spare2;
 ULONG Protect;         /*protection bits for this file*/
 ULONG Date[3];         /*date the file was created*/
 char FileName[36];     /*30 used for the file name*/
 ULONG Spare3[7];       /*all spares set to 0; reserved for future use*/
 ULONG HashChain;       /*header block for the next file with the*/
                        /*same hash value as this one*/
 ULONG Parent;          /*block where the parent directory resides*/
 ULONG Extension;       /*pointer to another block just like this*/
                        /*one, but which holds the next list of*/
                        /*data blocks for this file*/
 LONG Second;           /*secondary type for this block is -3*/
}
```

Data blocks for the FFS really have no structure, since they are entirely filled with data. For the OFS, there was a short structure:

```
struct FileDataBlock {
 LONG  Type;            /*For a data block, type is 8*/
 ULONG OwnKey;          /*the number of this same block*/
 ULONG SeqNumber;       /*sequence or ordinal number of this block in*/
                        /*the file*/
 ULONG DataSize;        /*how many bytes of data in this block?*/
 ULONG NextBlock;       /*which sector holds the next data block?*/
 LONG Checksum;
 char Data[488];        /*488 bytes of actual data*/
}
```


AMIGA **RGB CABLES**

CUSTOM RGB
AMIGA TO 4 BNC
SONY KV1311 23F-34F
SONY MULTISYNC 23F-9M
NEC MULTISYNC 23F-9M
MAGNAVOX ANALOG 23F-6MDIN
MITSUBISHI MULTISYNC 23F-25M
ATARI ANALOG ADAPTER 23F-13F

ALL Shipped Blue USA **29.00 TO 39.00**

Typical Non-Standard Cable

Call **For Quotes On Weird or Unusual Cables**

REDMOND CABLE

For The Dealer Nearest You!

EAST COAST 615-478-5760
WEST COAST 206-882-2009

68000 DISASSEMBLY

OTG Software **ANNOUNCING . . . DSM VERSION 1.0d**

DSM is a full-featured disassembler for the Amiga. Check out these features and you'll see why programmers agree, "DSM is the best disassembler currently available for the Amiga, bar none."

- DSM disassembles any Amiga program that does not use overlays.
- DSM produces assembler-ready output that is 100% compatible with the Amiga assembler, assem. Excellent results can also be obtained using the C.A.P.E. assembler from Inovatronics.
- DSM features true text detection.
- Improved disassembly is supported using the built-in expert system mode.

If your local Amiga dealer doesn't carry DSM, send check or money order to:

OTG Software
200 West 7th Street Suite 618
Fort Worth, TX 76102

TX residents add 7.25% sales tax. Price $67.50 US

TRY BEFORE YOU BUY!

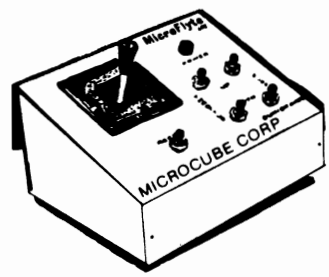
Best selling games, utilities and classics, plus new releases!

- 100's of titles
- Low prices
- Same day shipping
- FREE brochure

YES, WE ACCEPT:

RENT-A-DISC
Frederick Building #219
Huntington, WV 25701
(304) 529-3232

NOW AVAILABLE FOR THE AMIGA



The MicroFlyte JOYSTICK, the only fully proportional continuously variable joystick control for Flight Simulator II

". . .It transforms an excellent program into a truly realistic flight simulation system" B.A.C.E.

MICROCUBE PRODUCTS

Commodore 64/128
- MicroFlyte ATC Joystick **$59.95**
- Test/Calibration Disk: A diagnostic tool for your joystick **$4.95**

Amiga
- MicroFlyte Joystick—Plugs into the mouse port & works with most software . **$119.95**
- Analog Joystick **$74.95**

Include $4.00 shipping of joystick orders. FSII is a trademark of subLOGIC Corp.

Order Direct from:

MICROCUBE CORPORATION
P.O. Box 488
Leesburg, VA 22075
(703)777-7157

Hard Copy

A column on books and documentation

by Tim Grantham

Understanding Computers and Cognition
by Terry Winograd and Fernando Flores
Soft cover, 207 pages, 17.95 Cdn.
Addison-Wesley Publishing Co. Inc., 1987

I'm sure all of us who program have had the experience of racking our brains for hours over a bug, stretching reasoning powers to the limit to try to detect the logical flaw that's crashing the machine. Finally, exhausted, our minds in a hopeless whirl, we fall into bed - to wake the next morning with the answer as clear in our minds as the sun shining through the window.

Clearly, something happened while we were asleep, some kind of subconscious processing. I doubt that logical reasoning takes place: logic seems to be very much the province of the conscious mind, if *my* dreams are anything to go by. Yet, the answer to a logical problem involving some highly abstract symbols is there, fully realized. I've even had the experience of waking with the realization clear in my mind that there was a bug in a part of the program I hadn't thought about consciously at all.

The truth is that programming is essentially a creative endeavour. While it requires a disciplined mind to deal with the rigour of the logic, once a certain fluency with a computer language has been achieved, the process of programming becomes largely intuitive, rather than rational.

I remember reading a report of a study done at a university on what kind of people made the best programmers. Most people would probably say that the best programmers are people whose lives were ruled by logic: dispassionate, linear thinkers, with strong backgrounds in technology, who embrace the predictable, the rational; the spiteful would call them nerds or control freaks. Yet, to the surprise of everyone, including the authors, the study found that the best programmers were not scientists or engineers, but people with training in literature and music, the 'soft' studies. The authors could only offer the rather simplistic speculation that this was because both literature and music require creativity within highly formalized structures.

So it seems that, while we do use symbols in our thinking, our thinking, our intelligence, is not based on them. And that, say Terry Winograd and Fernando Flores, in their book *Understanding Computers and Cognition*, is why we will never have intelligent computers, nor even computers that can understand natural language, for computers, by their nature, are strictly manipulators of symbols.

Terry Winograd is one of the shining lights of computer science and artificial intelligence research, having worked and taught at MIT, Stanford University and the Xerox Palo Alto Research Center. He is the creator of SHRDLU, one of the first and best known experiments in artificial intelligence programming.

Fernando Flores was a cabinet minister in the government of Salvador Allende in Chile between 1970 and 1973. To quote from the introduction, "He was instrumental in a large-scale project to apply cybernetic theory to practical management problems." An odd combination of authors, perhaps, but one that has produced a book that ranges from Martin Heidegger's theories on understanding and being to the task of managing a chain of clothing shops.

In addition to the challenging statements referred to above, Winograd and Flores assert that: "Our central claim in this book is that the current theoretical discourse about computers is based on a misinterpretation of the nature of human cognition and language ... Computers designed on the basis of this misconception ... are restricted to representing knowledge as the acquisition and manipulation of facts, and communication as the transferring of information." (p. 78). The authors are at pains to point out that they do not intend to debunk so much as to reorient research into cognition and computers, "to create a new understanding of how to design computer tools suited to human use and human purposes." (p.8).

A crucial turning point for the authors in their own reassessment of cognition was the research by neurophysiologist Humberto Maturana and others into the physiology of vision. They found in their studies of frog vision that it was not the light intensity itself that stimulated the retinal nerves, but rather the pattern of local variation of intensity. For example, one type of fibre responded best to a small dark spot surrounded by light. When triggered, it led to activity appropriate for catching a fly in the location corresponding to the spot. In other words, at least some of the cognitive process of catching a fly was actu-

ally taking place in the frog's visual system, rather than in its brain. Furthermore, this cognitive process did not consist of the manipulation of internal symbols that stood for 'fly' and 'tongue' but was rather a concrete chain of chemical events.

Maturana went on to make use of a simple, well-known observation of colour perception to further question the relation between perception and the world being perceived. "When a stick is illuminated by a white light from one side and a red light from the other, it casts two shadows, one of which appears red (against a generally pink background) and the other of which appears *green*" (p. 41; the emphasis is the authors'). In the 'objective' reality here, there is no light from the band of wavelengths called green; yet the brain 'sees' green in the scene. The brain, in a sense, *creates* the world through its interpretation of its perceptions.

The authors meld Maturana's theories of the biological basis of cognition with Heidegger's philosophy of understanding and being and Hans-Georg Gadamer's study of hermeneutics (the study of the interpretation of texts, here broadened to include the interpretation of language) to challenge the rationalistic tradition in science, and particularly computer science.

This may sound daunting but Winograd and Flores are not attempting an exhaustive study. Their exposition is lively, with effective analogies and examples. In a discussion of the relevance of context to determining meaning, for example, the authors demonstrate the impossibility of establishing a context-independent meaning for such a seemingly simple term as 'water' with the following dialogue:

"A: Is there any water in the refrigerator?
"B:Yes.
"A: Where? I don't see it.
"B: In the cells of that eggplant.

"A claims that B's first response was a lie (or at best 'misleading'), while B contends that it was literally true. Most semantic theories in the rationalistic tradition provide formal grounds to support B, but a theory of language as a *human* phenomenon [my emphasis] needs to deal with the ground for A's complaint as well - i.e., with the 'infelicity' of B's reply.

"At first, It seems that it might be possible simply to expand the definition of 'water'. Perhaps there is a 'sense' of the word that means "water in its liquid phase in sufficient quantity to act as a fluid", so that a sentence containing the word 'water' is ambiguous as to whether it refers to this sense or to a sense dealing purely with chemical composition. But this doesn't help us in dealing with some other possible reponses of B:

"1. B: Yes, condensed on the bottom of the cooling coils.
"2. B: There's no water in the refrigerator, but there's some lemonade.
"3. B: Yes, there's a bottle of water in the refrigerator, with a little lemon in it to cover up the taste of the rust from the pipes." (p. 55-56)

While Winograd and Flores believe it is impossible to formalize meaning in conversation, they strongly believe it is possible to formalize the *structure* of conversation. 'Conversation' here is taken in the broader sense of 'interaction', as it might be found among the people in an organization, that may encompass many simple dialogues. They present a model of what they call "the basic conversation for action", a 'dance' of requests, declarations, promises and counters.

In the final chapters of the book, under the subtitle *A New Foundation for Design*, the authors describe how the model applies to the management of organizations and to the utilization of computer systems within those organizations. They describe a program they have developed that utilizes the power of a computer to enhance the effectiveness of such conversations. Winograd and Flores claim that the Coordinator Workgroup Productivity System, which runs on a network of workstations, is the first example of a new class of products that they have dubbed 'coordination systems' or 'workgroup systems'.

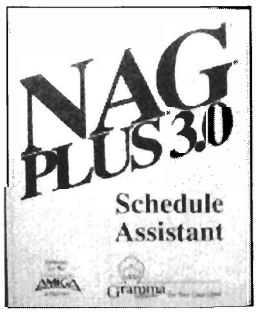
(A more detailed description by Winograd of the software can be found in the recent issue of *BYTE* magazine that focused on 'groupware'.)

The book concludes with the concept of the computer as "a structured dynamic communications medium". A computer, according to the authors, can never work within the realm of the *meaning* of the information it manipulates; it can work with the *structure* of that information, in such a manner as to transparently enhance the *user's* interaction with the meaning of the information. As an example, the authors refer to word processors, the best of which provide tools for the creation and editing of documents that are so effective that the tools quickly become invisible to the user.

The book covers an enormous amount of territory in a scant 179 pages. There is little room in a review to do justice to the ideas presented. Suffice it to say that the writing is accessible enough to have enabled me to read it on the streetcar on the way to work; yet the richness of the ideas made me pause every few pages, not in perplexity, but in wonder.

I'm not sure I agree with some of the statements made. Take "Nothing exists except through language". While I agree that objects do not exist until we separate them from their background, I'm not sure that this process is 'language'. I think that music and the human body, for example, *do* exist without language: they do not use symbols to communicate - they are what they communicate.

But whether or not you agree with what Winograd and Flores have to say, this provocative book will force you to re-evaluate everything you ever thought about computers and the way humans use them. It will make you question the nature of language, the nature of intelligence and the very nature of existence. As far as I'm concerned, *Understanding Computers and Cognition* is a must read.



NAG 3.0 PLUS Schedule Assistant
From Gramma Software
Suggested retail price: $79.95

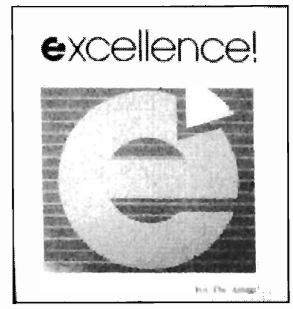
NAG 3.0 PLUS is an appointment scheduling program. That might not sound like the most exciting application, but Nag plus is fun to use, and is actually useful as an everyday tool. This is a thoroughly Amiga-oriented multitasking program; when not in use, it hangs around, ready to nag you about your appointments while you do other things with your Amiga.

Nagging you about appointments and events is, of course, Nag's basic game: you put them in the calendar, and you are "nagged" at predetermined times on or before the day or time arrives. There are also a host of related operations: you can also search for events, dial a number, print appointment lists, attach notes - and ARexx scripts! - to events. The scheme for entering appointments is particularly innovative, and the user interface, documentation, and online help all strike a near-perfect balance between simplicity and flexibility. As with most powerful programs, there is the initial bewilderment at the basic way things work, but that phase is passed quickly, thanks to the clear manual and the online help. The manual itself is very well designed: it first gives a quick explanation of using the program for the advanced user, then covers every operation individually in more basic terms.

Of course, there are those of us who never read the manual at all; that's where the program helps. Instant help is available for any gadget or operation just by clicking. The program is almost completely mouse-operated. For example, you select a day by clicking on it in a displayed calendar; you "check off" an appointment by clicking on it in a displayed list of events; you set appointment types and times by clicking on little arrows, etc. The basic operation is logical and simple, but since there are so many ways of setting appointments, there are a few tricks to learn. For example, you can set an appointment for one time, every year, every month, every second Tuesday in a month, etc. Picking the day from all of these choices only requires a few mouse clicks, once you learn how it works.

Nagging is done with a requester that pops up and displays the message you entered for that event, but you can also have accompanying speech and a sound effect. The Nag disk comes with dozens of amusing sound effects, for those that like a bit of whimsy to break up the day.

Overall, we recommend the program. If you are looking for an appointment scheduler, this is a good one. If you aren't, this may be one of those things you never thought you needed till you found you can't do without it.



excellence!
From Micro-Systems Software, Inc.
Suggested retail price: $299.95

Excellence! is a full-blown WYSIWYG word processor, with just about every feature you can think of: mail merge, spelling checker, thesaurus, grammar checker (!), automatic index and table of contents creation, proportional fonts, colour and graphics, PostScript output, multiple project windows, multiple columns, hyphenation, automatic footnotes, "glossary" macros, mathematical calculation, and probably a few things we haven't discovered yet. Being a true What-You-See-Is-What-You-Get affair, *excellence!* always shows your text in the format it will print, reformatting as you type, and when you change margins, line spacing, font, etc. It can handle different fonts, sizes, styles and colours within the same paragraph, along with pictures, if you wish.

All of this luxury can make writing easier, but it certainly doesn't make life easy for a program. As a result, *excellence!* pays a hefty price in size and performance. The program is over 340k on disk. Users with just 512k may have problems, since some of the features use additional memory. This is less of an issue for users with the recommended minimum RAM of 1 megabyte, but the performance issue is less easy to ignore. Entering text in the middle of a paragraph can be trying - an average typist can wait up to half a minute for a sentence to fully appear after it has been typed! This may not be a problem for some applications, but for creating a large document like a book, waiting for text to appear or for the formatted display to scroll might get frustrating when all you care about is creating and editing the text itself.

Of course, the philosophy of text-oriented versus WYSIWYG word processors is a religious issue, and no one is without their personal biases, including this reviewer. Philosophy aside, *excellence!* must be judged for what it is. It delivers on the features, it's well put together, it appears to be generally robust and thoroughly tested. The user interface is very easy to operate, and while there are plenty of features, they are easy to get at and use. The manual is top-notch in quality, well organized and generally helpful - and almost redundant, thanks to the slick user interface. Other than the performance question, there are very few bad things you can say about *excellence!*

Ah, but the performance question is a serious one. If you are a patient sort, this may not be an issue, but it seems to us that a $300 program shouldn't be frustrating to use. On the other hand, for all the on-screen formatting that takes place, you can almost think of *excellence!* as a simple desktop publishing program. And DTP text editing has never been fast.

Desktop Budget
From Gold Disk Inc.
Suggested retail Price: $69.95

The purpose of *Desktop Budget* is to help you plan where your money is going to be spent, and to keep track of how successfully you are sticking to the plan - in short, to build and maintain a household budget. Your first session or two with the program will be a construction phase, in which you distill your spending habits into a set of 'budget categories'. You can make these simple ('Salary', 'Food', 'Rent', 'The Rest') or detailed ('Inheritances', 'Lottery Winnings', 'Toothpaste', 'Cheeseburgers', etc.) as you prefer. You also set up 'accounts', which include bank accounts and credit cards; and 'assets', such as a house or a cottage, which have both a current value (figured into your 'Net Worth') and expenses (cost of upkeep).

From this point on, your interaction with the program consists of entering transactions - usually transferring money from an account to an expense category, or from an income category to an account - and generating reports, of which a wide variety is available. At year end, you do some cleaning up, then start over for the following year with a new budget.

If you are so far imagining *Desktop Budget* as some kind of merely functional, text-oriented adaptation of a standard big blueware package, you are very far astray. The most interesting aspect of the program is that the user interface to all of the functions described above is modelled very closely on the Amiga's Workbench - right down to the draggable icons, the 'Snapshot' menu option, and the Trashcan.

For instance: To perform a transaction, you double-click the icon for an account, which brings up a chequebook form. You enter the amount of the purchase and click an OKAY button: the cheque disappears and an icon for that transaction appears. You can then drag this over to the icon representing the expense category for the purchase. Subsequent reports will show the transaction, and the 'Actual Spending' monthly record for the budget category will be suitably updated.

Now, I'm not sure how this system compares for interaction speed with similar programs offering traditional user interfaces. It is definitely a lot more fun, and I'd guess that could make the difference between merely owning this kind of program and actually using it over the long haul.

Desktop Budget's very well planned and smoothly executed user interface is also worthy of attention from programmers who are looking for ways to give a fresh look to a musty application. There are plenty of good ideas here, and it would be surprising if they do not inspire other efforts in the same vein.

Schedule Assistant

Voice and Sound Reminders —
Create your own or use NAG Plus library.

Perpetual Calendar —
Enter 99 events per day.

Auto Dialer —
Connects YOU, not your modem.

AREXX Port —
Commands any timed event or action.

Notepad —
Click on any word to open text editor.

Print —
One click prints any event or appointment list.

AND MUCH MORE!

Suggested Retail $79.95. Ask your dealer or contact:

NAG® PLUS 3.0

Gramma software For Your Own Good
17730 15th Avenue N.E.
Suite 223
Seattle, Washington 98155
(206) 363-6417

Now For The Amiga!

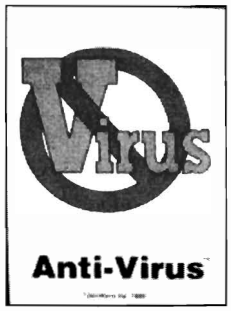
Are you tired of fumbling under or behind your computer to swap your mouse and joystick cables? Are your cable and computer connectors worn out from all the plugging and unplugging? Then Mouse Master is a must for you!

$39.95
plus shipping & handling.

(602) 322-6100

Practical Solutions®
1135 N. Jones Blvd., Tucson, AZ 85716

MOUSE MASTER



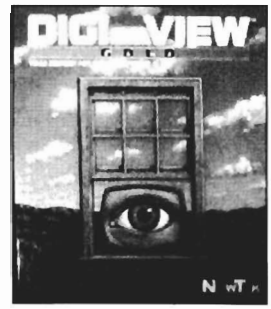
Anti-Virus by DevWare Inc.
Suggested retail price: $39.95

Anti-Virus is a collection of programs to help in the ongoing battle against viruses and "Trojan horse" programs. The main programs in the collection are vCat, vTrojan and vCheck. VCat is an easy-to-use, well designed application that combines a number of anti-virus defences. You can save and restore the boot block of any disk (where viruses live), protecting copy-protected software in case it ever becomes infected with a virus. The program also reports a non-standard boot block on a disk and lets you view the boot block, a sure-fire way to spot a virus. You can also install a special 'anti-virus' boot block, which displays a special full-screen graphic at boot time, indicating that the disk is virus-free. This is interesting from a technical viewpoint: the graphics appear almost instantly upon reboot, and it's a 640 by 400 picture with at least 16 colours - how do they get all of that in the boot block?

From a design standpoint, vCat is a gem: its large gadgets are totally self-explanatory, and everything works just as you'd expect. All the requesters are clear, and all potentially destructive operations provide a way out. User interface designers could learn a lot from a session with vCat.

VCheck is a virus monitor that remains active, displaying a tiny window in the Workbench screen that warns of disks with infected boot blocks. This is similar to public domain programs such as Steve Tibbett's VirusX. VTrojan checks the integrity of any number of files on your disk. You would typically use it to check CLI commands, libraries, fonts, or any other system files on the Workbench disk. Many "Trojan horse" programs spread destruction by infecting these files rather than boot blocks.

Our first reaction to a multiple-program anti-virus package was negative: it would be tempting to capitalize on the current virus panic by reworking some public domain programs and packaging them in a box that screamed of mass virus infection ruining your life. What we found instead was a well thought-out, logical package that we actually found useful: while testing vCat, we discovered a virus on a standard A2000 boot disk; using the anti-virus boot block helped us discover re-infection a few days later (the machine wasn't turned off before rebooting!). Perhaps soon the juvenile minds that create viruses will tire of their game, as soon as they realize that no one is impressed with their "cleverness". Until then, Anti-Virus is a useful addition to your software library, helping you with the detection, destruction, and prevention of computer viruses.



Digi-View Gold by NewTek, Inc.
Suggested retail price: $199.95

Digi-View Gold is the latest version of NewTek's popular video digitizing system. The package consists of the digitizing hardware (a small box that plugs into the Amiga's parallel port), version 3.0 of the Digi-View software, a manual, and a filter wheel. All you need is a black and white camera (a stand comes in handy, too), and you have everything you need to capture any image from a photo or object and turn it into a standard IFF format picture.

Basic operation is the same as always: you digitize red, green and blue images separately, using a different filter over the camera lens each time to create colour images; black and white images can be produced with a single scan. The main difference from version 2.0 of the software is in the support of overscan (up to 768 by 480) and 'halfbrite' mode, and a new 'line art' mode for diagrams. The hardware looks all-new, and is claimed to produce better results than the previous design.

The way the software manipulates the cumbersome HAM mode to produce clear and colourful images is close to miraculous; even in 320 by 200 resolution, it's hard to see individual pixels, and there's no evidence of the fuzziness and 'fringing' often found in HAM pictures. Using dithering to mix new colours and carefully assigning colour registers in HAM mode, the software does everything it can to create the best possible image.

Digi-View's best work, however, can only be done with a little help from the operator. Actually digitizing the subject is only half the job; the next step is manipulating the program's colour controls to get the best image. Internally, the digitized image is stored with 21 bits of colour information per pixel, giving the program plenty of data to work with. By sliding around controls for brightness, contrast, saturation, red, green, blue, and sharpness, you can drastically alter the look of the picture, and reveal detail that you didn't even think was there. You can also select the number of colours used in the final image; the best is the 4096+ enhanced HAM mode, but it is useful to be able to choose 32 colours or less to display the image on other screens. You can even define the colour palette yourself, and Digi-View will map the image to those colours! This is great for integrating digitized images with your own software, or for merging several pictures together on one non-HAM screen.

The image digitizing and processing capabilities of Digi-View Gold are tremendous for the price. The best images are produced with some effort on the part of the user, but the reward is some of the best images possible on the Amiga's screen.

News LINK

Zorro II Prototyping Boards For Hardware Developers: Celestial Systems has announced a prototyping board designed especially for the Amiga 2000 computer and other members of the Amiga family with Zorro II expansion slots. The board allows developers, engineers, and dedicated hobbyists to build, test, and debug custom hardware before incurring the expense of developing a (possibly faulty) custom PC board. The two-sided board has over 4400 plated-through holes on a 0.1" grid. The main prototyping area accepts DIP ICs with up to 64 pins, and PGAs (Pin Grid Arrays) as large as 14 by 14 pins. The I/O connector pattern and included mounting bracket are designed for standard 'D' connectors with up to 37 pins. The prototyping board and mounting bracket are available directly from Celestial Systems. The retail price, in US funds, is $49.95. Contact Celestial Systems at 2175 Agate Court, Simi Valley, CA 93065-1839. Phone 805/582-0729.

The MediaPhile Infrared Control System from Interactive Microsystems: The *MediaPhile* system provides complete interactive control of video decks, laser and compact disk players, and other infrared-controlled devices from Amiga computers. Learner/transmitter software lets you teach the Amiga infrared commands from any remote control unit. The *MediaPhile* controller plugs into the second mouse port and transmits infrared commands from a three-foot LED extension cable. Two Sony S-port control cables, a remote pause control cable, and two video deck counter inputs are also provided.

Programmer's toolkit software is provided as a standard AmigaDOS sharable library for developers who wish to control infrared devices from their application programs. In addition to being callable from virtually any compiled language, an *ARexx* query entry point provides acces to the *MediaPhile* software library from any *ARexx* program or function host. FD and BMAP files are included for *AmigaBasic* users. The controller also operates with Interactive MicroSystems' *MediaProcessor* video editing software and *MediaBase* Interactive sales system. The system is $495 and can be purchased directly from Interactive MicroSystems, Inc., 80 Merrimack St., PO Box 1446, Haverhill, MA 01831. Phone 508/372-0400.

Amiga Desktop Video Production System: The *MediaPhile* system has been expanded to include control of Super VHS, VHS, 8mm, and ED Beta video decks. Edit video tape from a player to a recorder deck, overlay computer graphics and titles, and record animation frames automatically with the system's new *MediaProcessor* software. Configured with two Sony EV-S800 8mm decks, the *MediaPhile* system records a database of 48 hours of digital stereo sound or 8 hours of video. Play back hundreds of audio and video selection automatically from one or two decks. One deck seeks the start of the next selection while the other plays.

The *MediaPhile* controller reads infrared commands from the remote control units of video decks, laser disk players and other infrared devices and stores them on disk. *MediaProcessor* and other application software using the *MediaPhile* Programmer's tookit library send the stored infrared commands through the controller's transmitter. See previous item for contact information.

The Disc Company launches its one million dollar software acquisition campaign: The Disk Company has unveiled its plans to invest over $1,000,000 during 1989 to acquire innovative software products for the Commodore Amiga.

"The Disk company is aggressively seeking new products that offer distinct benefits to our customers and that will differentiate our company from the competition" says A. Robert Kotick, TDC President. "We are offering independent developers an attractive compensation package consisting of generous advances and significant royalties, and because of our financial resources and marketing talent we expect to attract a number of noteworthy submissions."

TDC will launch an international advertising campaign in all the major Amiga periodicals which details the ways in which independent software artists can benefit from TDC's worldwide publishing network. In addition, TDC will offer existing small software publishers an opportunity to participate in its Affiliated Label program - through which small publishers can gain access to international markets.

Of particular interest are audio, video, graphics and entertainment software. Programmers whose software shows special promise will receive in-depth technical assistance and the financial support necessary to bring the product to market. For more information, contact Jack Edelstein, 3135 South State St., Ann Arbor, MI 48108. Phone 313/665-5540.

Superplan merges spreadsheet, project management, business graphics: Precision Incorporated, makers of the *SuperBase* family of data management software, have announced *SuperPlan*, a new integrated, ARexx-compatible system for Amiga business users that combines a spreadsheet, project management software and a business graphics system into a single, mouse-driven environment. In addition to a 74-function spreadsheet, *Superplan* comes with a macro language that lets anyone tailor their working environment, allowing the creation of turnkey business applications. *Superplan* allows project management activities, including critical path analysis, time/expense planning, resource allocation, gantt and PERT charts. Business graphics capabilities let you choose from over 18 different kinds of charts. Full ARexx support allows for free exchange of data with other programs, including *Super-*

Base Professional 3, Precision's powerful database. *Superplan*'s suggested retail is $149.95 (US). For more information, contact Precision Incorporated, 8404 Sterling St., Suite A, Irving, TX 75063. Phone 214/929-4888.

Mindware announces fast, flexible 3-D rendering package: Mindware of Ontario, Canada has announced the release of *PAGErender 3D* for the Amiga, a three-dimensional rendering and animation system with a large object library and full support for stereo vision - using either Haitex's X-Specs glasses or red/blue stereo vision glasses (which are included). In addition to a large library of polyhedrons, letters, numbers and more, *PAGErender 3D* comes with tools for building and restructuring your own objects into cones, crevices, domes, waves, etc. Objects can be made to accelerate, decelerate or move along any straight or abitrary path. Objects can spin around any specified axes of rotation. *PAGErender 3D* also reduces the time needed to generate animations, by automatically generating its own animation scripts while you work, which can be edited later. Features include ray tracing, adustable dither and light intensity, four different draw modes in each resolution and the ability to create scenes that are larger than display size. *PAGErender 3D* sells for $159.95 and works on all Amigas with 512k or more. Contact: Mindware International, 110 Dunlop St. W., Box 22158 Barrie, Ont., Canada L4M 5R3. Phone 705/737-5998 or call 1-800-461-5441 toll-free.

SupraDrive hard disk systems for the A2000: The SupraDrive for the A2000 is a high-performance hard disk system that features autobooting, fast access time, and Supra's software and technical support. It comes in 20, 30 and 60MB capacities that retail for $699, $799, and $995, respectively. The SupraDrive features true DMA access and an external SCSI port that allows connecting addition SCSI devices to the system. The SupraDrive is compatible with all hardware, and supports MS-DOS partitions when a Bridgeboard is used. Both internal and external units are available. The internal model comes with everything needed to mount it in an Amiga 2000, including step-by-step instructions. The external unit connects via an interface card that plugs into an expansion slot. Special interface kits are also available for those who already have a hard disk that they plan to mount internally. An interface with a controller is $399.95, or $249.95 without controller. Each kit comes with formatting and utilities software, including the program CLImate. SupraDrive owners can obtain assistance from technical support staff. Contact Supra Corp. at 1133 Commercial Way, Albany, OR 97321. Phone 503/967-9075.

We have had a SupraDrive on evaluation at the Transactor offices for some time. We recommend this system: it is very easy to install, coming with all parts required, and excellent instructions and set-up software. It is also very reliable, unlike other hard drive systems we've tried on the 2000; used constantly in magazine production, it has never crashed or failed in any way. The drive itself is slightly noisier than some other units, but not excessively so. In our experience, the SupraDrive system is a better choice than a Commodore 2090 controller card with an off-the-shelf drive unit.

From Progressive Peripherals & Software: Dr. Term Professional is a terminal program that features the following: • Capable of Amiga to mainframe communication • 300 entry phonebook • Each phonebook entry may have custom communication parameters • Extensive command language for programming function key macros • Script language • Remote mode with multiple-level password acts like a mini-BBS • Emulation capabilities include VT100, VT52, TTY, and Dr. Term Pro • Mouse or keyboard-driven commands • Extensive manual • latest in communication protocols, including Xmodem, Ymodem, and WXmodem. Price is $99.95.

Math-Amation Mathematical Quantity Processor: A series of mathematical tools that are powerful and easy to use. Covers unit conversion, statistics, algebra, graphics, geometry, and calculus. Features include: • Business graphics module • IFF format support for graphics • Memory window keeps track of previous results, making them available for calculations • Programmable scientific calculator • Matrix calculator • Geometric calculations • Polynomials including integration, differentiation and plotting, etc., Function plotting on functions of one variable; two functions may be plotted and points of intersection located using the zoom facility • statistical analysis on a column of data • curve fitting includes Line, Log, Exp and Polynomial of degreee 'n'. Price is $99.95.

ProGEN Genlock: • Meets RS-170A standards • No colour saturation loss • No chroma phase shift • BNC Connectors for true Genlock quality • Parallel chromatic processing based on custom VLSI chip technology • Eight adjustable points provide compatibility with all professional equipment • Composite colour video output for *all* Amigas • Will allow video transfer to VCR • Stand alone unit allows convenient placement • Comes complete with hardware and software. Price is $399.95

DiskMaster is a file utility system for the Amiga. The mouse-driven interface lets you move, copy and rename files, make directories and create multiple file copies, ARC and de-ARC files • Two large directory displays let you view over 70 files at once • Define up to 12 physical or logical devices for quick access • Full support for most archive utilities, including ARC, Zoo and PAK • View IFF pictures, play IFF sounds or instruments, and print text files • Mouse-controlled file reader. Price is $59.95.

Superbase professional database management system: *SuperBase Professional* consists of a forms editor, a text editor, a report generator and a database management language with over 200 commands. The forms editor lets you create screen forms that are fully relational and fully interactive: they can perform cross-file validation, calculation and data checking. Within a form, you can cut and paste fonts, boxes and graphics images. The text editor is a word processor that lets you integrate text blocks of any size into your database and use them like ordinary text fields. The report generator performs relational searches in various files and merges results with the forms editor for easy presentation. The database management language lets you automate all mouse/menu operations

and customize the entire interface, including creating your own pull-down menus and requesters. The language is based on BASIC and lets you incorporate password protection and file encryption. All the features of *Superbase* are incorporated into *Superbase Professional,* including the VCR-style control panel, picture database capability, unlimited file size, etc. Price is $349.95.

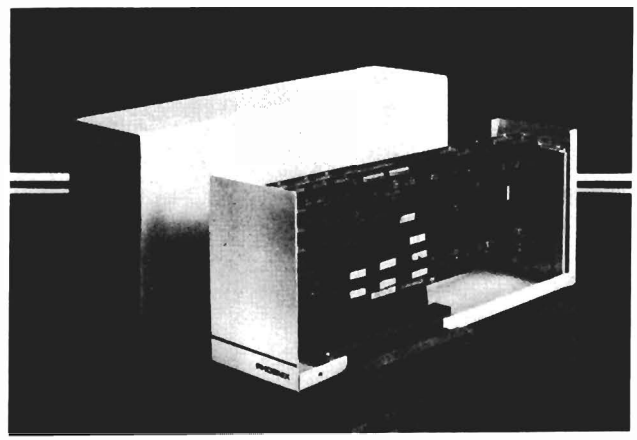
For information on the above products, contact Progressive Peripherals & Software, Inc. 464 Kalamath St., Denver, CO 80204. Phone 303/825-4144 FAX 303/893-6938.

A-Talk III from OXXI Inc: The latest version of A-Talk has added Zmodem, Ymodem and Ymodem batch, an expanded script language as well as an ARexx interface. Emulations supported are VT100, VT52, H19, ANSI, TTY and Tektronix 4010/4014. Graphics can be captured in IFF and Aegis Draw format for export to graphics programs, allowing downloading of graphics from mainframes. The expanded script language has over 50 commands with a learn mode for easy creation of scripts. List price is $99. Registered owners of A-Talk Plus can upgrade for $25; MaxiComm owners can migrate for $50; Diga! and Online! trade-ins are available for $60. (Add $4 shipping and 6.5% in California to all orders.) Order from: Oxxi, PO Box 90309, Long Beach, CA 90809 or call 213/427-1227 for further information.

ASDG plugs Amiga into world's first hand-held scanner: The Sharp JX-100 colour scanner can be used with any Amiga, using Scanlab/100, a new hardware/software combination from ASDG Incorporated. The JX-100 can scan an area of approximately 4" by 6" at 200 DPI. Scanlab/100 is the latest addition to ASDGS's Professional Scanlab family of systems that allow compatibility between the Amiga and Sharp's JX-100, JX-300, JX-450 and JX-600 line of colour scanners. The JX-100 will sell for $995. Pricing for Scanlab/100 has yet to be announced. Contact ASDG Incorporated, 925 Stewart St. Madison, WI 53713. Phone 608/273-6585.

DevPac Amiga Version 2: Hisoft has released an all-new version of the acclaimed DevPac assembly language development system. It includes an integrated screen editor, assembler, and debugger. The assembler runs at a maximum speed of 75,000 lines per minute and includes conditional assembly, macros, disk inclusion, and a full expression evaluator. It is fully compatible with the Commodore assembler. The debugger has dynamic, multiple breakpoints, single-step search facilities, and a multi-window display. Low-cost upgrades are available to users of version 1. Contact Michtron, 576 South Telegraph, Pontiac, MI 48053. Phone 313/334-5700 FAX 313/334-3553.

The Phoenix Expansion Chassis from Phoenix Electronics Inc. is an expansion chassis for the Amiga 500 or 1000 that runs all major A2000 cards. There are two models available for both the A500 and 1000, one including an internal power supply, and one without. Planned for future release is a third model including a coprocessor slot with bus buffering and arbitration. Internal power supplies supply enough power to run hard cards and up to 8 megs of RAM. For pricing and ordering information, contact Phoenix Electronics Inc., P.O. Box 156, 314 Court St., Clay Center, Kansas 67432. Phone 913/632-2159, FAX 913/632-5382.



Designer Database series from Software Visions is a ready-to-use collection of useful database macros. These off-the-shelf applications are designed to work with Software Visions' Microfiche Filer and Microfiche Filer Plus database engines. The first two disks in the series, to be released on March 31, are the Home I and Business I disks. The Home I disk includes applications such as videotape, audio recording, stamp and recipe catalogues, a home budget, personal inventory, and more. The Business I disk includes a daily calendar, mail merge (with macros), expense report, general ledger, an invoicing/inventory application, and more. The Home I disk is $39, and Business I is $59. Soon to follow will be video/graphics and sound I, Home II, and Business II. Contact Software Visions at PO Box 3319, Framingham, MA 01701. Phone 1-800-527-7014 or 508/875-1238.

Aegis announces interactive music disks: Aegis Development, Inc. has announced the release of *Sonix SoundTrax Volumes One & Two*, a series of high-quality interactive songs for *Aegis Sonix*. Written and performed by Jon Rami, these disks offer the ultimate sound clarity by using clean sampled instruments. Each song has keys carefully set on the Amiga keyboard, making an off-key note almost impossible. The *SoundTrax Volumes*, each containing two full disks of songs, will be available in January at a suggested retail price of $19.95 each. Contact Aegis Development at 2115 Pico Blvd., Santa Monica, CA 90405. Phone 213/392-9972, FAX 213/392-6445.

Broadcast-quality Genlock from VidTech: The Scanlock VSL-1 allows Amiga users to integrate processed references signals, such as input from a video camera or a VCR, with the

Amiga's video signals. The Scanlock VSL-1 is available immediately and retails for $995. The European version (PAL) sells for $1,095. The VSL-1's front panel is equipped with independent fade controls for reference and computer video. In Y/C mode, the Scanlock provides a key-out signal that is fully synchronized to the reference video. The fully synchronized signal eliminates colour dot crawl on reference or computer video and can be used by an external chroma keyer. Sync in the output video follows sync in the reference video, even when computer video is created using a different interlace mode. VidTech also announced receiving a 3.5 million dollar contract for the delivery of several thousand Scanlock units from Southern Technologies, Inc. of Dallas, Texas. Contact: VidTech International, Inc., 2822 NW 79th Ave., Miami, FL 33122. Phone 305/477-2228 FAX 305/591-1651.

Amiga touch screen from Amigo Business Computers: Future Touch is a touch screen that is compatible with all Amiga software, and provides touch input for all mouse functions. Developers can also design programs specifically for touch screen applications or include touch options in their programs. *SuperBase Professional*, for example, allows the user to designate touch sensitive areas on the screen for use in database programs. The 13 inch version of *Future Touch*, with 640 by 200 resolution, is offered as a complete touch screen monitor which retails for $1595, or as a kit for retrofitting into existing Amiga monitors for $1195. The 19 inch version, with 640 by 480 resolution, comes fully assembled for $3390.

Amigo Comports is a multi-port RS-232 communications card for the A2000 that auto-configs and comes with two to eight ports; prices start at $389. Amigo Comports is available from Amiga dealers or direct from Amigo Business Computers.

Amigo BackPack is a cartridge tape backup system for the Amiga. It provides 40 megabytes of backup on 3M DC-2000 1/4 inch "mini-tapes", with special software to provide inherent redundancy and error elimination during the backup process. Amigo Backpack comes in two versions: the internal device, with a suggested retail of $899, fits in the 5 1/4 inch disk slot. The external device, at $949, plugs into the last disk drive on the external drive chain and configures automatically. For more information on the above products, contact Amigo Business Computers at 192 Laurel Road, East Northport, New York 11731. Phone 516/757-7234.

New Gold Disk products: Desktop Budget is an easy to use personal finance management program. Intuitive in operation, it is icon-based and works like the familiar Workbench. Desktop Budget assists in setting up monthly payments and automatically performs month-end reconciliations and year-end rollovers. It also includes a calculator and icon editor. Suggested retail is $69.95, in the U.S. and Canada.

Transcript is billed as an "incredibly fast writing/editing tool". It is a word processor designed to keep up with your typing speed no matter how large the document. Transcript's features include a 90,000 word spelling checker, automatic index generation, mail merge and automatic hyphenation. Suggested retail is $69.95, in the U.S. and Canada.

300 DPI Black & White Scanner: Gold Disk has developed a hardware/software interface for the Canon IX12 black and white 300 dot-per-inch scanner. This scanner will allow the user to save images as black and white IFF files at 75, 150, 200 or 300 dots per inch. Software control allows the user to define regions of scanned pages for high-resolution use. The scanned images can be used in any program supporting the IFF standard, like Deluxe Paint II, Professional Page, or Professional Draw. The Canon IX12 scanner complete with parallel port hardware and software interface is scheduled for release in March 1989 at a suggested retail of $1095 U.S. For information on the above products, Contact Gold Disk at:

PO Box 789, Streetsville
Mississauga, Ontario
Canada L5M 2C2.

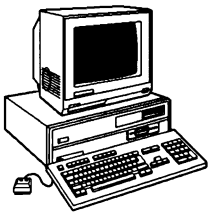
Phone 416/828-0913
FAX 416/828-7754.

AMIGA Experts

Speak with the people who use the *Amiga*.. Our crew has over 30 years combined experience with the *Amiga*. !
All of the latest Software & Hardware in stock.

We ship via UPS Mon.-Fri.

Commodore authorized repair, low flat rate plus parts.
Warranty service
Store Hours:
Mon.-Thur. 10 - 6
Friday 10 - 8
Saturday 9 - 5



The Memory Location

396 Washington Street
Wellesley, MA 02181
(617) 237-6846
(617) 237-6440

10 % Discount on Software to all club & user group members!

Reader Service Card

Fill out this card (please print) or attach your peel-off mailing label, and circle the reader service number for the ads that you'd like more information about (see Advertiser Index below).

Name _____

Company _____

Address _____

_____

Postal/Zip Code _____

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

TA V2I1

Advertiser	Page	Reader Service #
Antic Software	37	1
ASDG. Inc.	27	2
Briwall	32-33	3
Comp-U-Save	17	4
Compplications	16	5
Creative Computers	12-13	6
D-Five Associates	63	7
Discovery Software	Outside Back Cover	8
Eagle Tree Software	47	9
Electronics 2001	63	10
Express-Way Software Inc.	47	11
Fisher's Computers	63	12
Fuller Computer Systems	5	13
Gramma Software	75	14
Great Valley Products	Inside Back Cover	15
Guru's Guide	11	16
Inovatronics Inc.	1	17
Intercomputing Inc.	21	18
Interface Technologies	7	19
Lake Forest Logic Inc.	31	20
Lattice Inc.	Inside Front Cover	21
MCP Associates Inc.	63	22
Metadigm	56	23
Micro Systems International	53	24
Microcube Corporation	71	25
Micromaster Inc.	53	26
Microsmiths Inc.	41	27
OTG Software	71	28
Practical Solutions	75	29
Redmond Cable	71	30
Rick Stiles (UEdit)	5	31
Safe Harbour Computer Supplies	27	32
Software Excitement	65	33
The Memory Location	80	34
William S. Hawes	5	35

for the AMIGA Transactor

The Magazine for Amiga Programmers

Please print your name and address below (or transfer your mailing label)

Check here for change of address ☐

Name: _____

Address: _____

City: _____ *Province/State:* _____

Zip/Postal: _____ *Phone:* _____ *CompuServe ID:* _____

NOTE: Prepayment required. Purchase orders will be accepted ONLY if accompanied by payment.

Subscribers only: please enter subscription number from mailing label ☐

Subscription Order Section

PLEASE NOTE THAT TRANSACTOR FOR THE AMIGA IS NOW PUBLISHED MONTHLY

	Canada ($Cdn.)			U.S.A. ($US)			Foreign ($US)			Air Mail ($US, overseas only)		
	Mag	Disk*	Both*	Mag	Disk	Both	Mag	Disk	Both	Mag	Disk	Both
12 issues	☐35.00	☐123.00	☐158.00	☐27.50	☐101.00	☐128.50	☐38.50	☐119.50	☐157.50	☐ 73.00	☐ 145.00	☐ 218.00
24 issues	☐65.00	☐220.00	☐270.00	☐49.00	☐160.00	☐190.00	☐69.00	☐220.00	☐290.00	☐ 135.00	☐ 250.00	☐ 390.00

* Ontario residents please add 8% of disk subscription price for provincial sales tax. No PST on magazine subscriptions.

☐ **Please send Microfiche instead of magazines** ☐ **Send me info on Transactor Classic (C64/128)**

Combo Subscription Bonus!

If you are subscribing to BOTH the magazine and the disk, check one of the following FREE gifts:

☐ **Transactor Classic Amiga Disk #1** ☐ **Fish Catalog Disk** ☐ **Extended Subscription (13 issues)**

Product Order Section

Ontario residents please add 8% PST on all items

$Cdn.	$U.S.	
☐ 14.95	12.95	Transactor Classic Amiga Disk #1
☐ 14.95	12.95	Transactor Classic Amiga Disk #2

Fish Disks $6.95 US, $7.95 Cdn. each

Note: Disk numbers not yet released may be ordered in advance - simply add numbers to end of list. Disks 57, 80 and 88 are not available.

☐ 1	☐ 2	☐ 3	☐ 4	☐ 5	☐ 6	☐ 7	☐ 8	☐ 9	☐ 10	☐ 11	☐ 12	☐ 13	☐ 14
☐ 15	☐ 16	☐ 17	☐ 18	☐ 19	☐ 20	☐ 21	☐ 22	☐ 23	☐ 24	☐ 25	☐ 26	☐ 27	☐ 28
☐ 29	☐ 30	☐ 31	☐ 32	☐ 33	☐ 34	☐ 35	☐ 36	☐ 37	☐ 38	☐ 39	☐ 40	☐ 41	☐ 42
☐ 43	☐ 44	☐ 45	☐ 46	☐ 47	☐ 48	☐ 49	☐ 50	☐ 51	☐ 52	☐ 53	☐ 54	☐ 55	☐ 56
☐ 58	☐ 59	☐ 60	☐ 61	☐ 62	☐ 63	☐ 64	☐ 65	☐ 66	☐ 67	☐ 68	☐ 69	☐ 70	☐ 71
☐ 72	☐ 73	☐ 74	☐ 75	☐ 76	☐ 77	☐ 78	☐ 79	☐ 81	☐ 82	☐ 83	☐ 84	☐ 85	☐ 86
☐ 87	☐ 89	☐ 90	☐ 91	☐ 92	☐ 93	☐ 94	☐ 95	☐ 96	☐ 97	☐ 98	☐ 99	☐ 100	☐ 101
☐ 102	☐ 103	☐ 104	☐ 105	☐ 106	☐ 107	☐ 108	☐ 109	☐ 110	☐ 111	☐ 112	☐ 113	☐ 114	☐ 115
☐ 116	☐ 117	☐ 118	☐ 119	☐ 120	☐ 121	☐ 122	☐ 123	☐ 124	☐ 125	☐ 126	☐ 127	☐ 128	☐ 129
☐ 130	☐ 131	☐ 133	☐ 134	☐ 135	☐ 136	☐ 137	☐ 138	☐ 139	☐ 140	☐ 141	☐ 142	☐ 143	☐ 144
☐ 145	☐ 146	☐ 147	☐ 148	☐ 149	☐ 150	☐ 151	☐ 152	☐ 153	☐ 154	☐ 155	☐ 156	☐ 157	☐ 158
☐ 159	☐ 160	☐ 161	☐ 162	☐ 163	☐ 164	☐ 165	☐ 166	☐ 167	☐ 168	☐ 169	☐ 170	☐ 171	☐ 172

The Transactor for the *AMIGA* Disk $12.95 US, $14.95 Cdn. each

(these disks correspond to issues of **Transactor for the *AMIGA***)

☐ Vol. 1, Iss. 1 ☐ Vol. 1, Iss. 2 ☐ Vol. 1, Iss. 3 ☐ Vol. 1, Iss. 4 ☐ Vol. 1, Iss. 5 ☐ Vol. 1, Iss. 6

☐ Vol. 2, Iss. 1

Add flat $2.00 shipping & handling for any number of items ordered from above box.

Back Issues, Microfiche: $4.50 U.S., $4.50 Cdn. (MF=Microfiche, filled box=SOLD OUT)

☐V1#1 ☐MF ■V1#2 ☐MF ☐V1#3 ☐MF ☐V1#4 ☐MF ☐V1#5 ☐MF ☐V1#6 ☐MF

☐V2#1 ☐MF

Ontario residents please add 8% provincial sales tax for any items ordered from this box.

☐ Visa ☐ MasterCard ☐ Cheque/MO enclosed Cheque #_____ Dated ____/____/____ Amount _____

Acct. # _____ Expires ____/____ Signature _____

Please allow 6-8 weeks for delivery in the U.S., 4-6 weeks in Canada.

FOLD ALONG SCORES WITH CORRECT REPLY CARD ON OUTSIDE. USE TAPE, PLEASE - NO STAPLES OR GLUE.

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 390 BUFFALO, NY
POSTAGE WILL BE PAID BY ADDRESSEE

Transactor
P.O. Box 338 Station 'C'
Buffalo, NY, 14209-9990

place stamp here

Transactor
10-85 West Wilmot Street
Richmond Hill, Ontario, Canada
L4B 1K7

Use our convenient...

TOLL-FREE ORDER LINE
1-800-248-2719 Extension 911

Here's a convenient way to order magazine or disk subscriptions and other products. Nothing to fill out or drop in the mail! Just call the toll-free number above and have your credit card number ready. Available in the United States only.

NOTE: The toll-free number can be used for ordering only; for subscription queries, etc., call or write to our office directly.

Business Reply Mail
No Postage Stamp Necessary
If mailed in Canada

Postage will be paid by:

Canada Post Postes Canada 46487

Transactor
501 Alden Road
P.O. Box 3250
Markham Industrial Park
Markham, Ontario
L3R 9Z9

Advertising in Transactor for the Amiga

Interested in advertising in *TransAmi?* Call Julie Cale at 416/764-5274

AMIGA 500 USERS!
NOW GET A2000 POWER!!

Announcing

THE WORLD'S FIRST AUTOBOOTING HARD DISK FOR THE AMIGA 500

IMPACT A500 – HD/RAM SUBSYSTEM

offers everything any A500 owner has ever dreamed of – in one compact, easy-to-install, add-on subsystem; provides the same high performance as our top-of-the-line A2000 controller

IMPACT™ A500 – HD/20
A500 – HD/30
A500 – HD/40Q
A500 – HD/45
A500 – HD/80Q

Available with a built-in 20, 30, 40, 45 or 80 Mega Byte Hard Drive and an optional 2MB plug-in FAST RAM/AUTOBOOT expansion module.

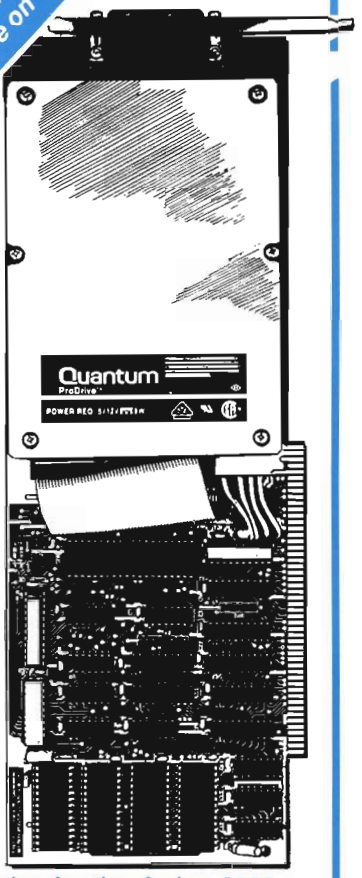
- AUTOBOOT directly from hard disk
- Connect up to 6 SCSI peripherals
- Internal & external SCSI connections
- DMA data transfer to onboard 16K buffer
- All partitions can be FFS with V1.3 AUTOBOOT
- 11/19ms average access time on 40Q & 80Q models

A2000 USERS!

Still

THE MOST POWERFUL HARD DISK SOLUTION FOR YOUR A2000!

IMPACT™ A2000 - HC/30
A2000 - HC/40Q
A2000 - HC/45
A2000 - HC/80Q
A2000 - HC/100



The GVP IMPACT-HC "Harddisk-on-a-Card" series for the Amiga 2000 matched with the 3.5" Quantum ProDrive hard disk drives, provide turbo performance and unbeatable value at list prices as low as $18.00 per Maga Byte! The Quantum ProDrives provide average access times as low as 11ms with their built-in 64KB read-ahead cache and offer a MTBF of 50,000 hours which is almost twice the industry average.

Now Shipping

AUTOBOOT EPROMS

GVP

GREAT VALLEY PRODUCTS INC.

225 PLANK AVENUE, PAOLI, PA 19301 TEL (215) 889-9411 FAX (215) 889-9416

SWORD OF SODAN™

Grasp the Sword of Sodan and feel the true power of arcade action!

Creaking, groaning, shrieking,
teeth clenching arcade realism.
Deep dark forests.
Misty, eerie graveyards.
Treacherous, guard-filled streets.
Dare to enter the evil castle,
Craggamoore.
Wield your sword through
its winding corridors
and rank rotting dungeons.
Your story will be
the greatest ever told
— Your reward will be
the greatest ever known!

FEATURES:

- ***Battle with massive animated characters that stand 2/3rds the size of the screen.***
- ***Fight as Hero or Heroine through 11 incredibly challenging levels!***
- ***Study your past performance in Replay Mode! (extra memory required)***

Packed into 3 disks with more than 4 MEGABYTES of detailed graphics and digitized sounds!

Available for Amiga™, Coming soon for IBM® and compatibles, Apple II GS.

AMIGA screens shown.

DISCOVERY SOFTWARE

Fight your way to your local software retailers. Or, call 1-800-342-6442 for VISA and MasterCard Orders. For check or money orders, send $49.95 plus $3 shipping and handling to DISCOVERY SOFTWARE INTERNATIONAL INC. 163 Conduit Street, Annapolis, MD 21401. Allow 4-6 weeks for delivery. Sword of Sodan is a trademark of Discovery Software International, Inc. Amiga is a trademark of Commodore Amiga, Inc. IBM is a registered trademark of International Business Machines, Inc.

Dorian 1988 ©